Shader composition in Use.GPU

https://usegpu.live

Steven Wittens - 2024/10/02

WESL Community

Topics

- Origin of Use.GPU
- **Examples**

- Problems with GPU dispatch
- **WGSL extensions**
- **Data structures**

- Q&A

Topics

- Origin of Use.GPU
- **Examples**

- Problems with GPU dispatch
- **WGSL extensions**
- **Data structures**

- Q&A

Problem-Solution Ordering

Topics

- Origin of Use.GPU
- **Examples**

- Problems with GPU dispatch
- **WGSL extensions**
- **Data structures**

- Q&A

Problem-Solution Ordering

Player must find
locked door
before key

The problem with GPUs is
that **in your first week,**
they **teach you wrong…**

The problem with GPUs is that **in your first week,** they **teach you wrong…**

…and it takes **years** to **unlearn that.**

The problem with GPUs is that **in your first week,** they **teach you wrong…**

…and it takes **years** to **unlearn that.**

What do they **teach you**?

The problem with GPUs is that **in your first week,** they **teach you wrong…**

…and it takes **years** to **unlearn that.**

What do they **teach you**?

How they did it **15 years ago.**

I could just show you the **shaders**...

I could just show you the **shaders**...

...or I could show you why you'll **end up in the same place** in 3 years.

Eric Arnebäck @erkaman2 · Sep 10, 2017

Computer graphics is a field of study where your main objective is to solve this research question: "why the #$@& is my window black!?!"

12 179 651

Eric Arnebäck @erkaman2 · Sep 10, 2017

Computer graphics is a field of study where your main objective is to solve this research question: "why the #$@& is my window black!?!"

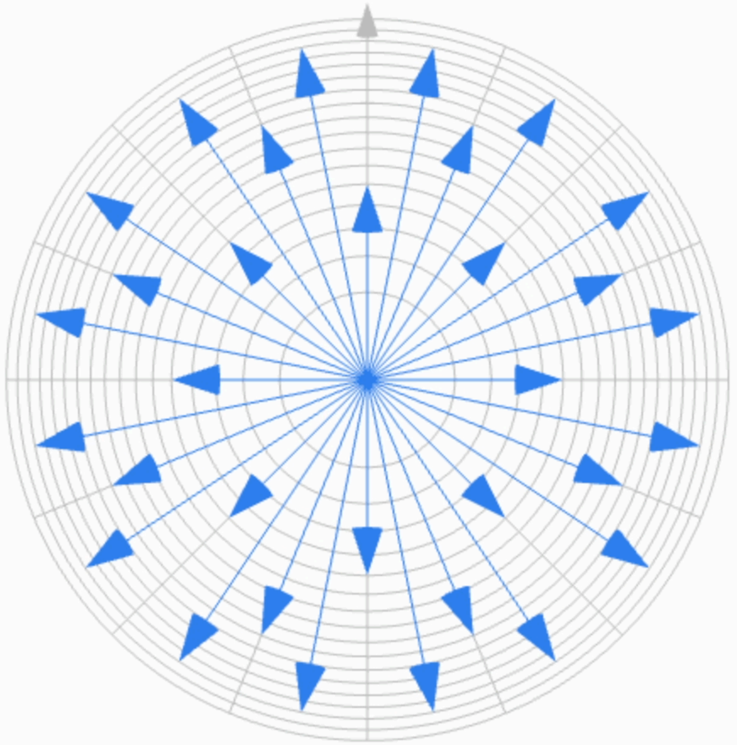
12 179 651

Ryan Schmidt @rms80 · Jun 6

a great thing about being in the non-rendering part of computer graphics is that you will always have great job options, because 95% of the industry is obsessed with just rewriting the **same** renderers over and over

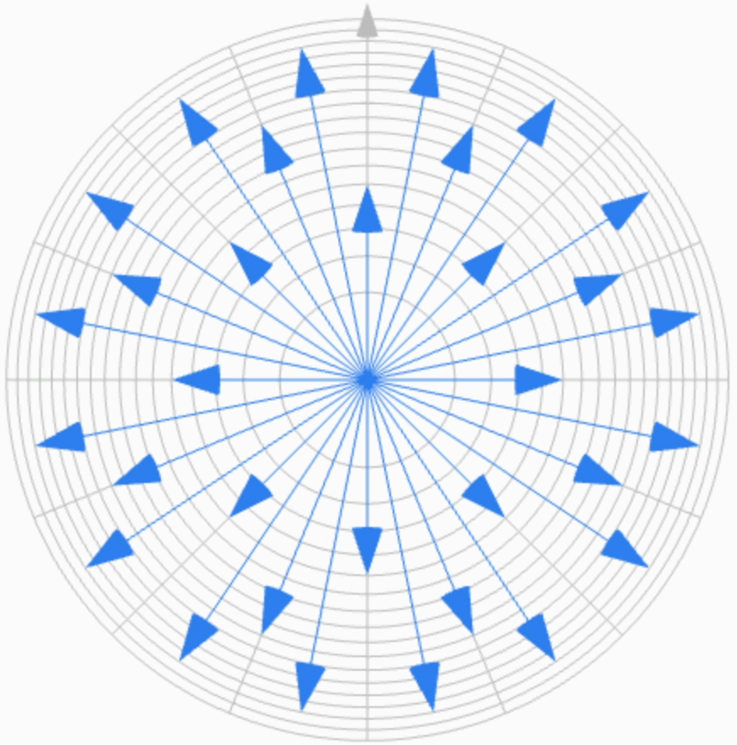
8 7 96 9.6K

something new. Just like the negative numbers complemented the positives, and the fractions snugly filled the space between them—and the reals somehow fit in between *that*—we need to go look for new numbers where there currently aren't any.



Thus the square root actually looks like this. New numbers flow in from the 'far side' as we try and shear the disc apart. The complex plane is stubborn and wants to stay connected, and will fold and unfold to ensure this is always the case. This is one of its most remarkable properties.

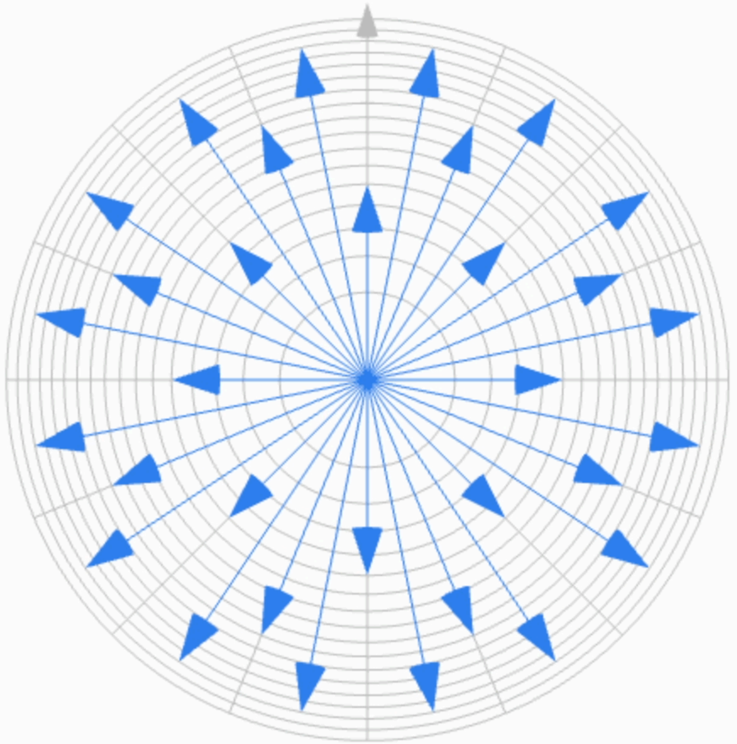
something new. Just like the negative numbers complemented the positives, and the fractions snugly filled the space between them—and the reals somehow fit in between *that*—we need to go look for new numbers where there currently aren't any.



Thus the square root actually looks like this. New numbers flow in from the 'far side' as we try and shear the disc apart. The complex plane is stubborn and wants to stay connected, and will fold and unfold to ensure this is always the case. This is one of its most remarkable properties.

something new. Just like the negative numbers complemented the positives, and the
ed the space between them—and the reals somehow fit in
eed to go look for new numbers where there currently aren't any.

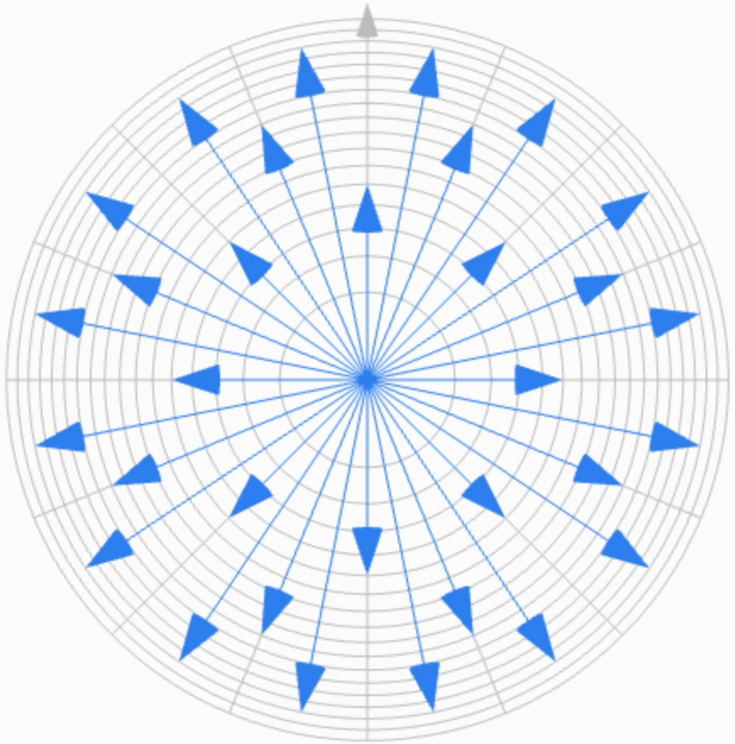
Where it all started...
How to Fold a Julia Fractal (2013)



Thus the square root actually looks like this. New numbers flow in from the 'far side' as we try and shear the disc apart. The complex plane is stubborn and wants to stay connected, and will fold and unfold to ensure this is always the case. This is one of its most remarkable properties.

Where it all started...
How to Fold a Julia Fractal (2013)

something new. Just like the negative numbers complemented the positives, and the
ed the space between them—and the reals somehow fit in
eed to go look for new numbers where there currently aren't any.



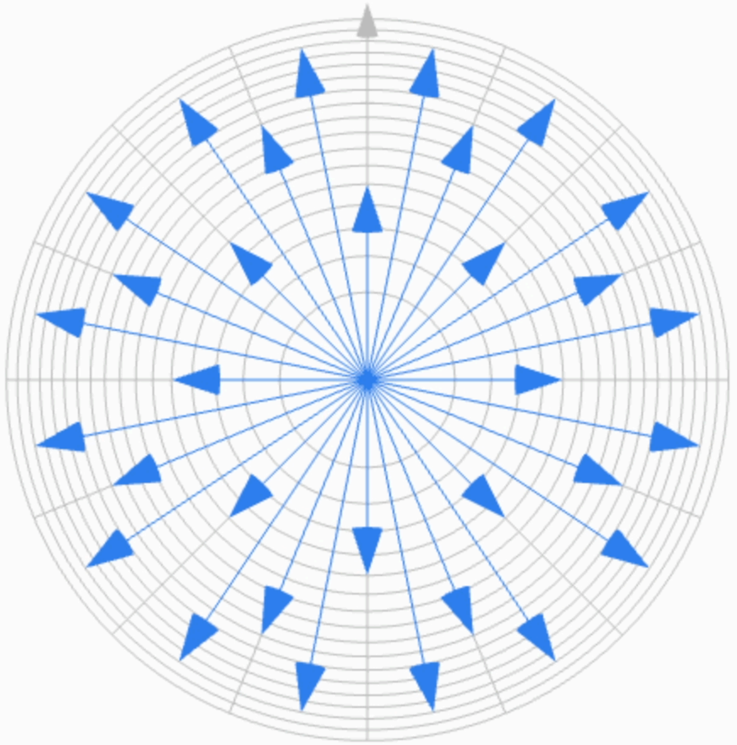
Thus the square root actually looks like this. New numbers flow in from the
disc apart. The complex plane is stubborn and wants to stay connected, an
this is always the case. This is one of its most remarkable properties.

something new. Just like the negative numbers complemented the positives, and the

ed the space between them—and the reals somehow fit in

eed to go look for new numbers where there currently aren't any.

Where it all started...
How to Fold a Julia Fractal (2013)



"How do I draw a f😿ing line???"

v numbers flow in from the

ants to stay connected, an

this is always the case. This is one of its most remarkable properties.

WebGL 3D Geometry

Today

Today

Use.GPU

Today

Use.GPU

WebGPU

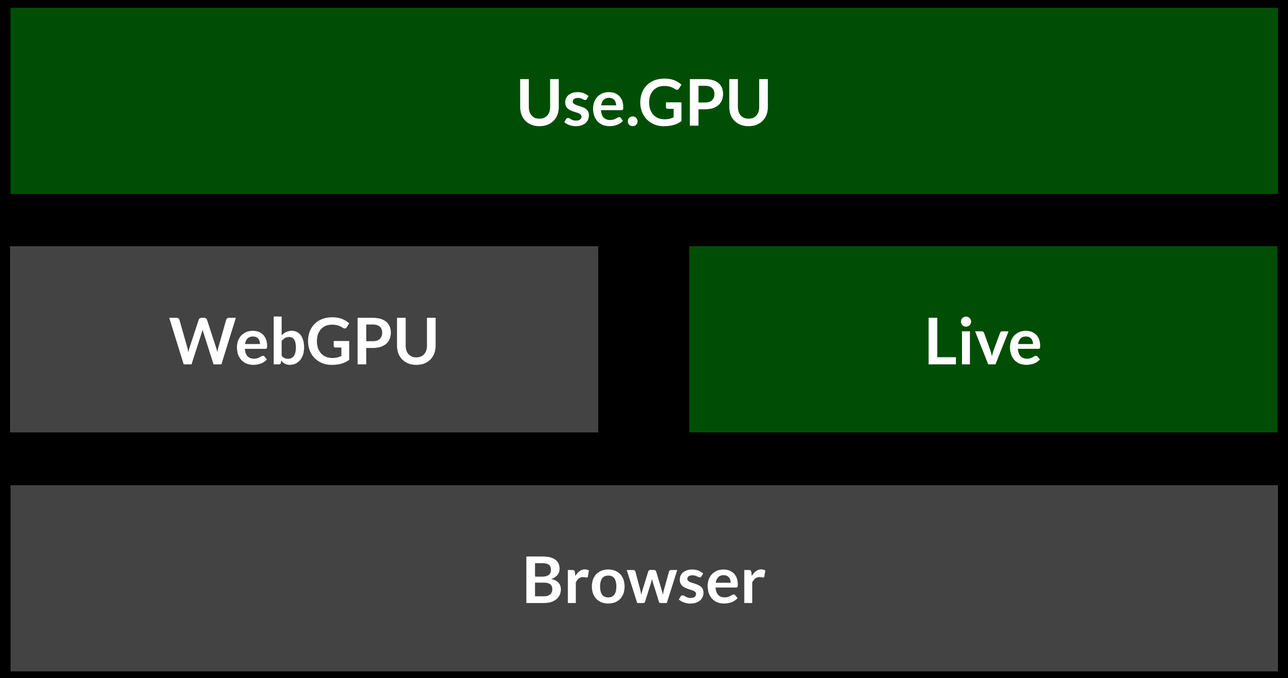
Today

Use.GPU

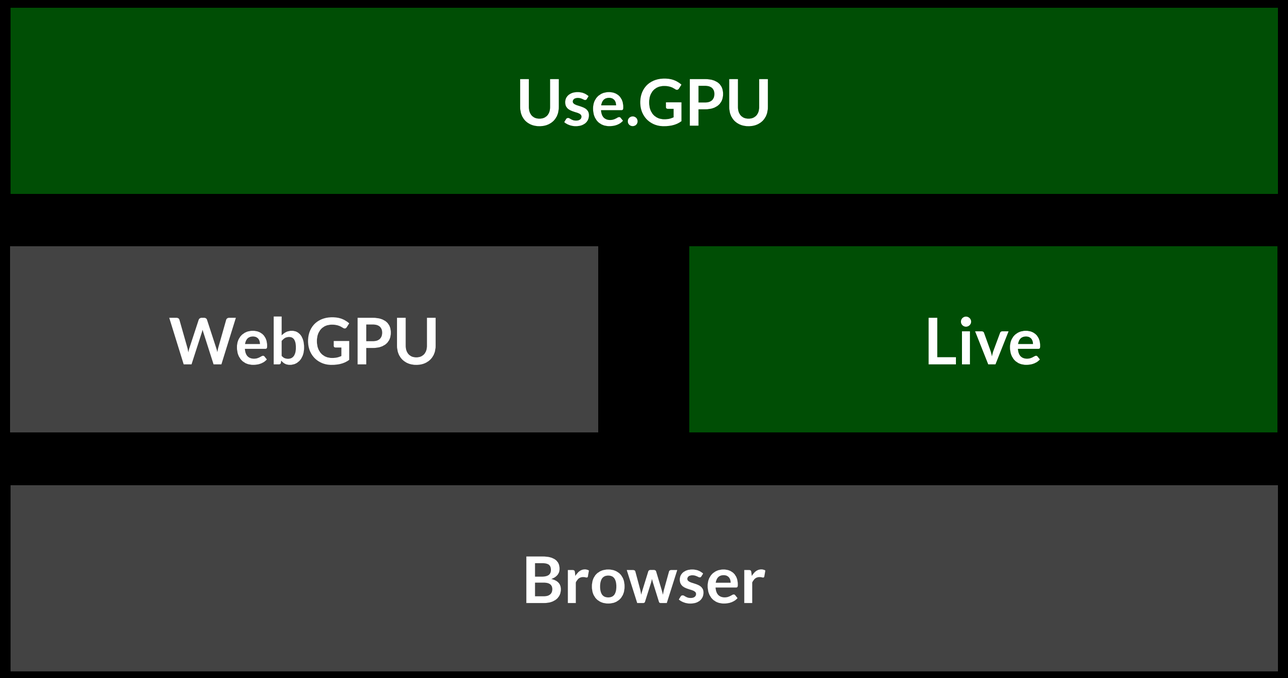
WebGPU

Live

Today

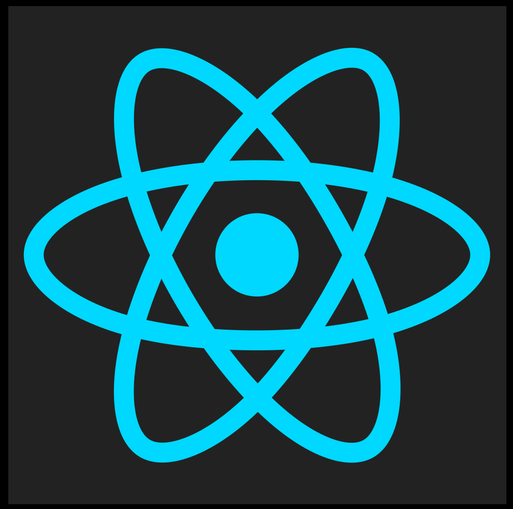


Today



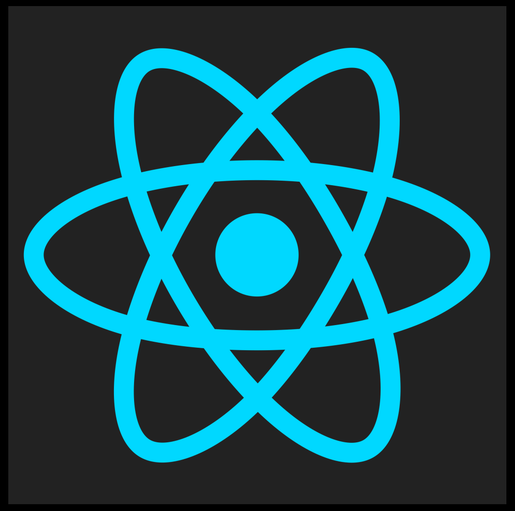
Use.GPU is an architecture of **necessity**

Live = alt-React



Live = alt-React

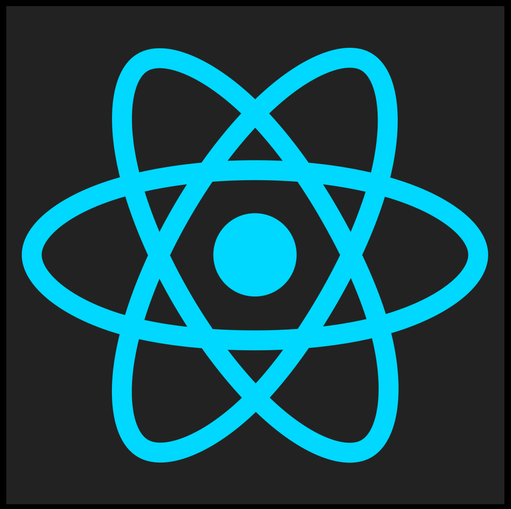
React clone



Live = alt-React

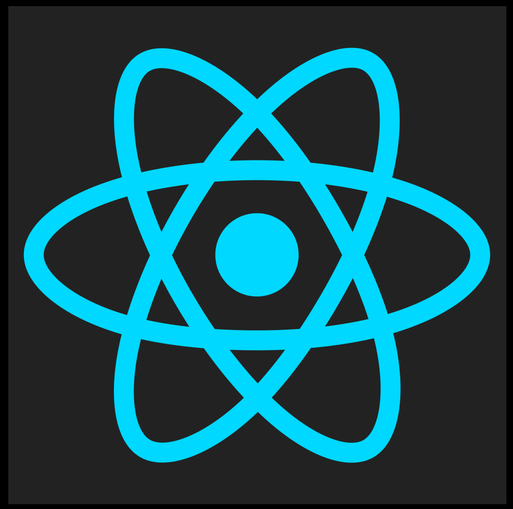
React clone

Needed to manage the **shader bureaucracy**



Live = alt-React

React clone



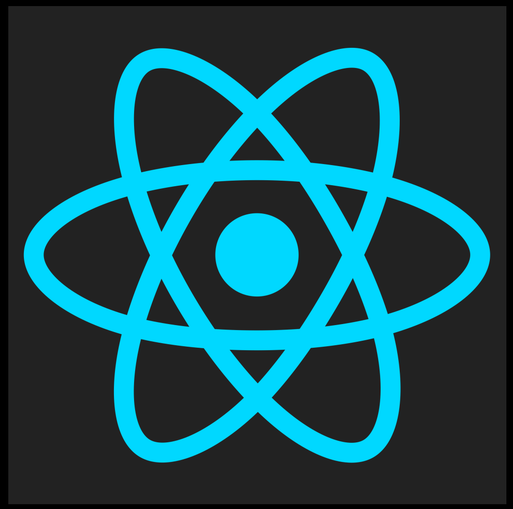
Needed to manage the **shader bureaucracy**

- **Incremental**
  Avoid redundant recomputation

Live = alt-React

React clone

Needed to manage the **shader bureaucracy**

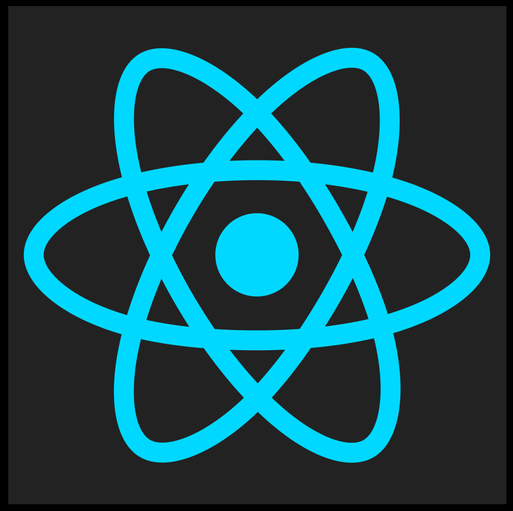


- **Incremental**
  Avoid redundant recomputation
- **Reactive**
  Dispatch and data flow is implicit and 1-way

Live = alt-React

React clone

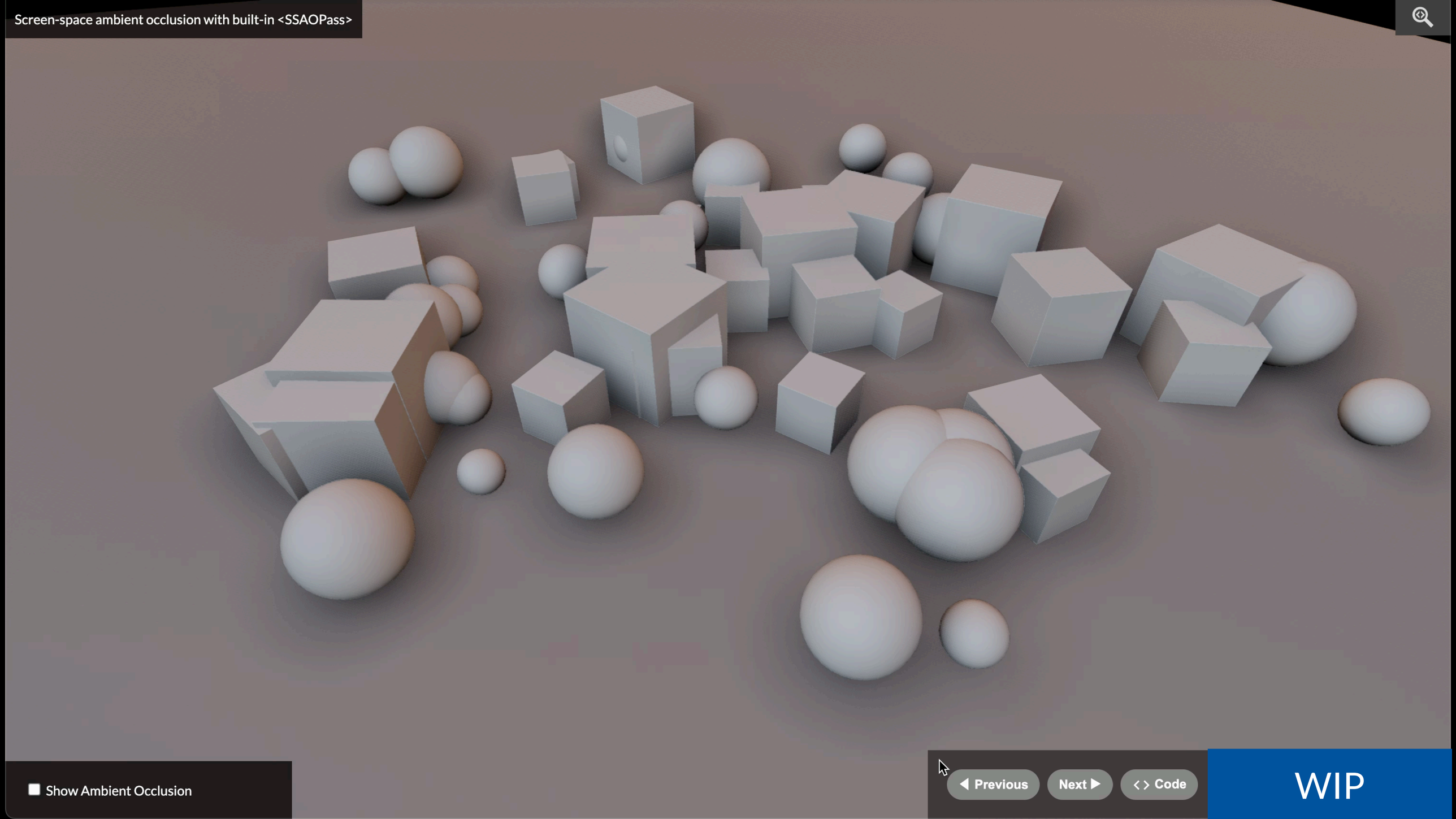
Needed to manage the **shader bureaucracy**



- **Incremental**
  Avoid redundant recomputation
- **Reactive**
  Dispatch and data flow is implicit and 1-way
- **Declarative**
  Side-effects are auto-mounted and disposed

Examples

Screen-space ambient occlusion with built-in <SSAOPass>

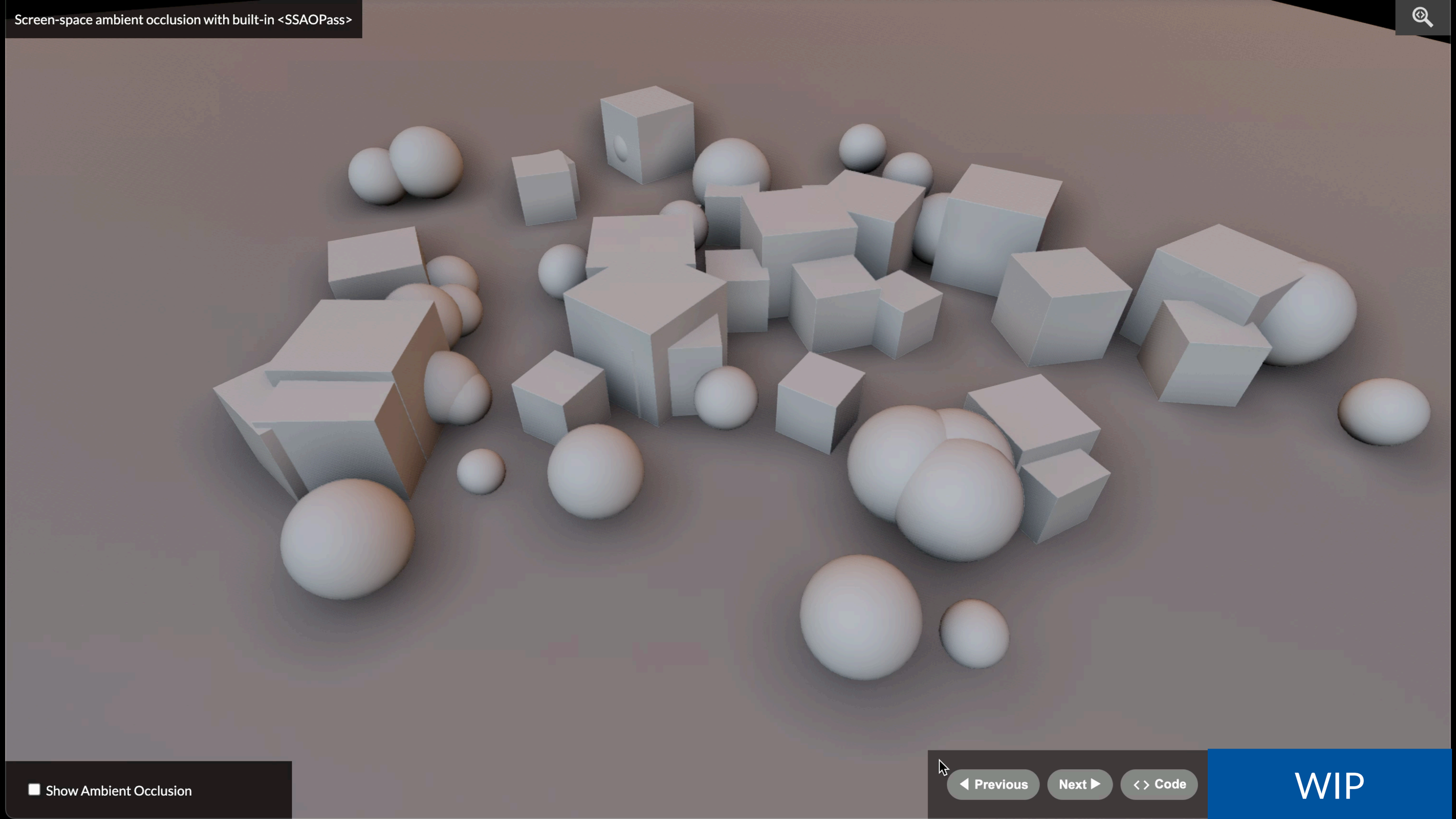


Show Ambient Occlusion

◀ Previous Next ▶ <> Code

WIP

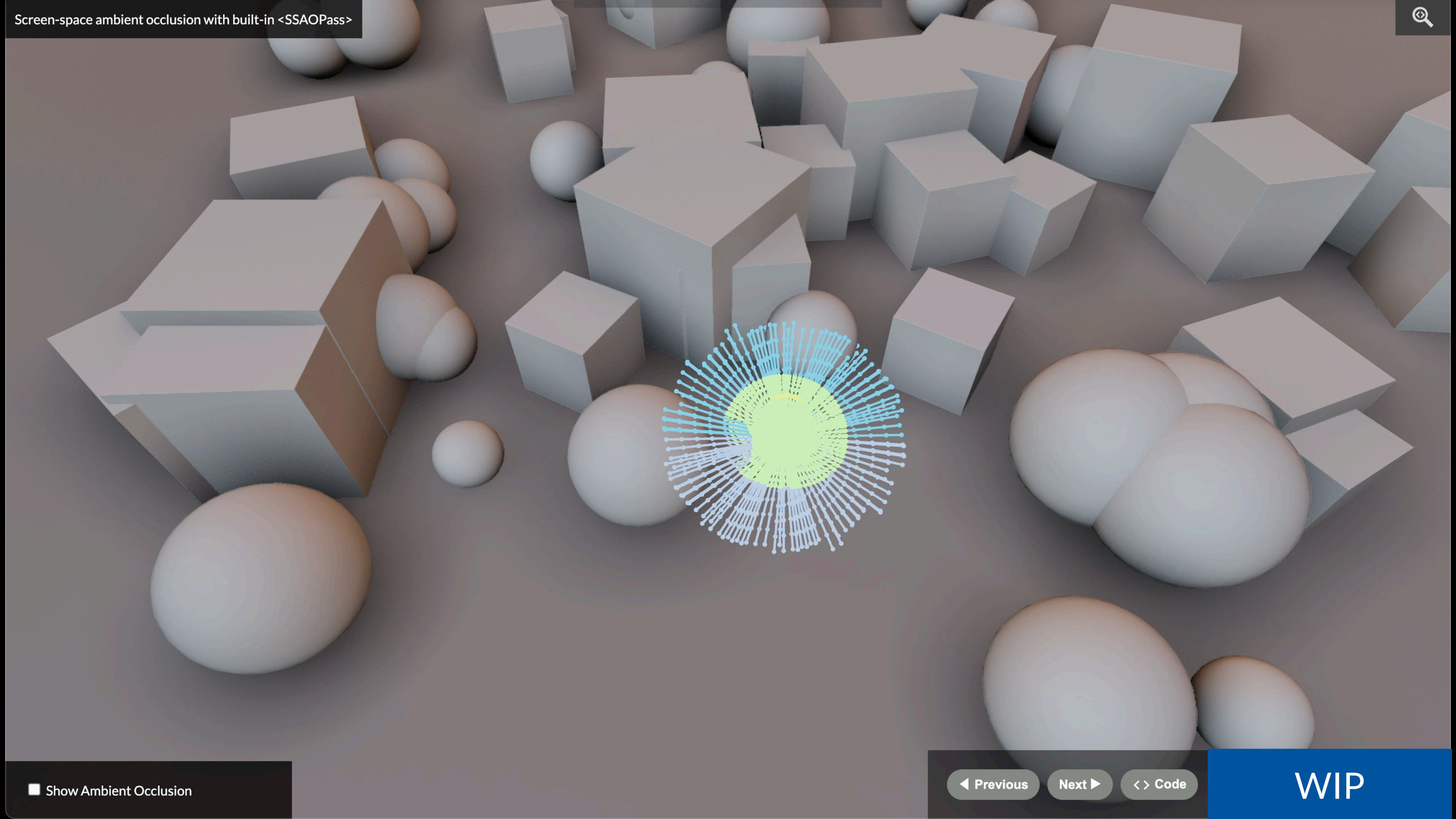
Screen-space ambient occlusion with built-in <SSAOPass>

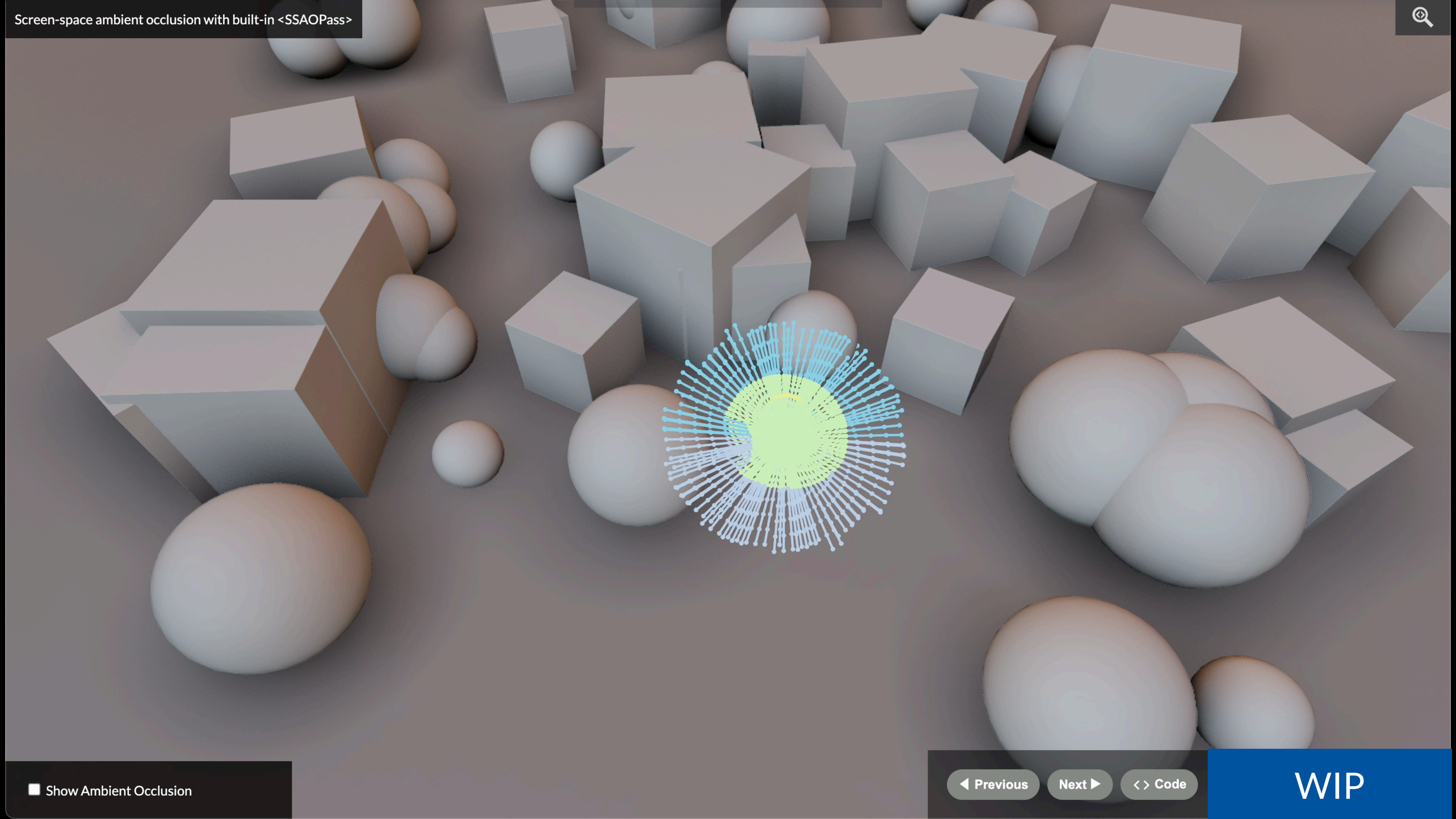


Show Ambient Occlusion

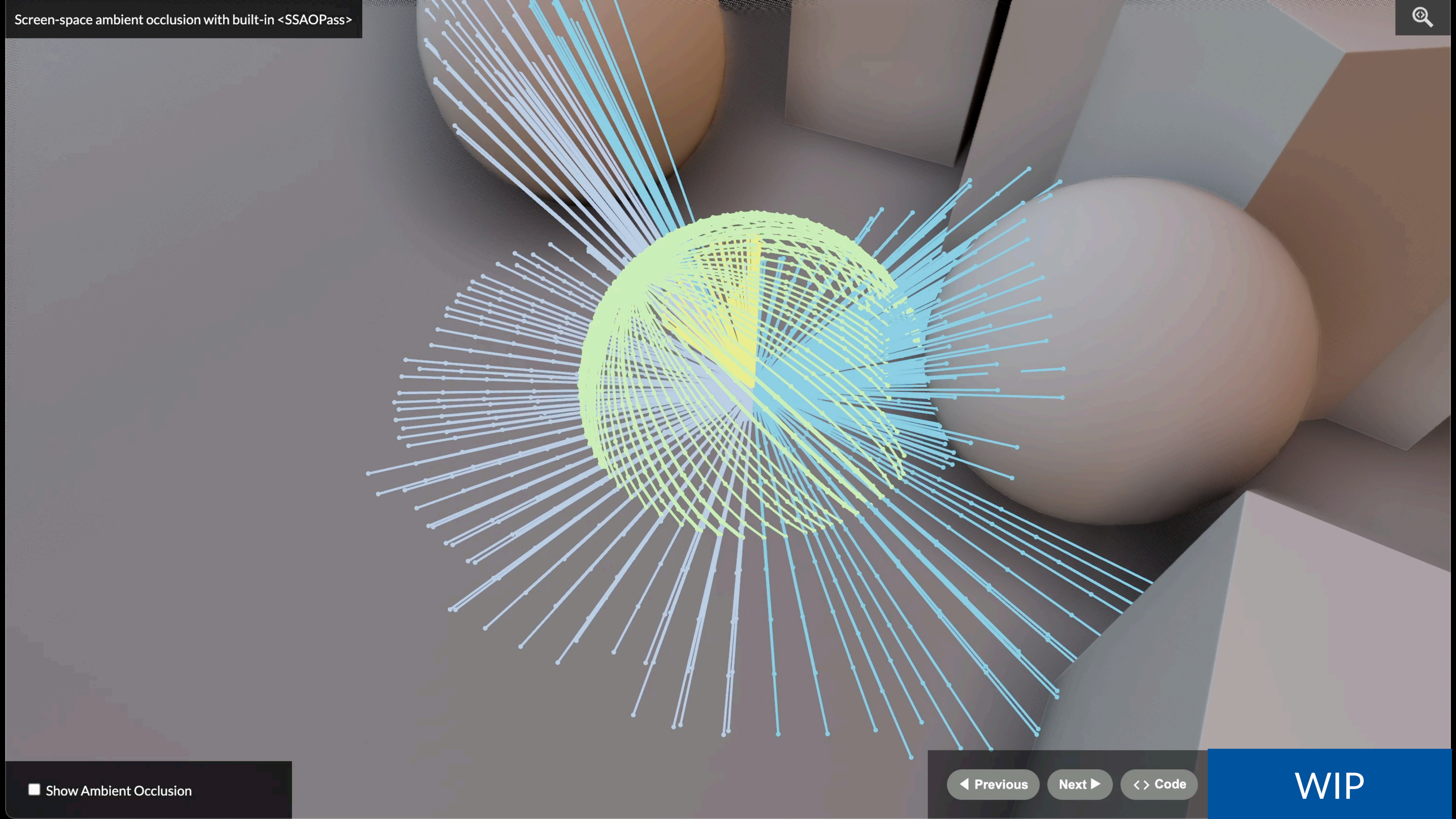
◀ Previous Next ▶ <> Code

WIP


Screen-space ambient occlusion with built-in <SSAOPass>
Show Ambient Occlusion
Previous
Next
Code
WIP


Screen-space ambient occlusion with built-in <SSAOPass>
Show Ambient Occlusion
Previous
Next
Code
WIP

Screen-space ambient occlusion with built-in <SSAOPass>

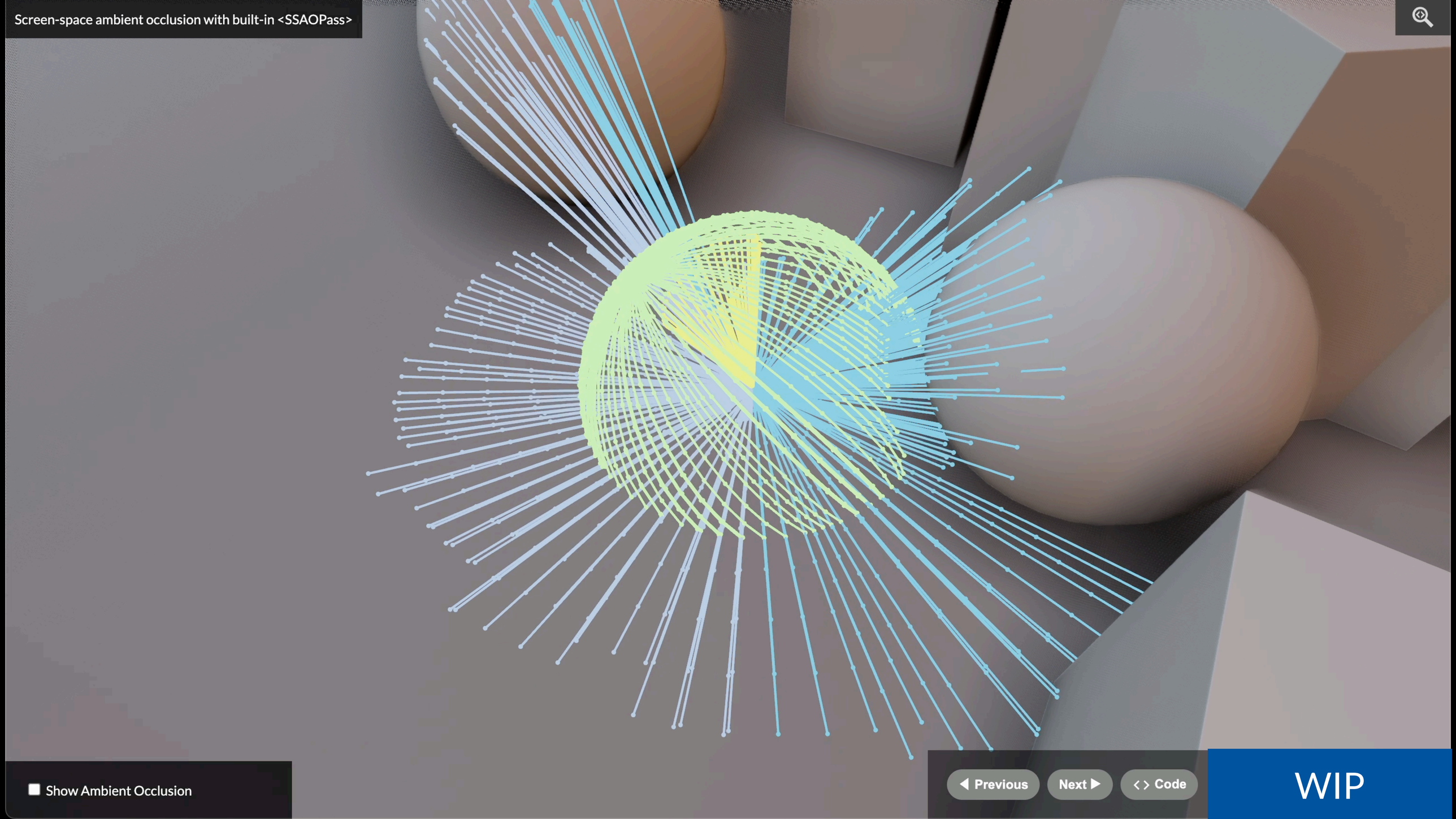


Show Ambient Occlusion

◀ Previous Next ▶ < > Code

WIP

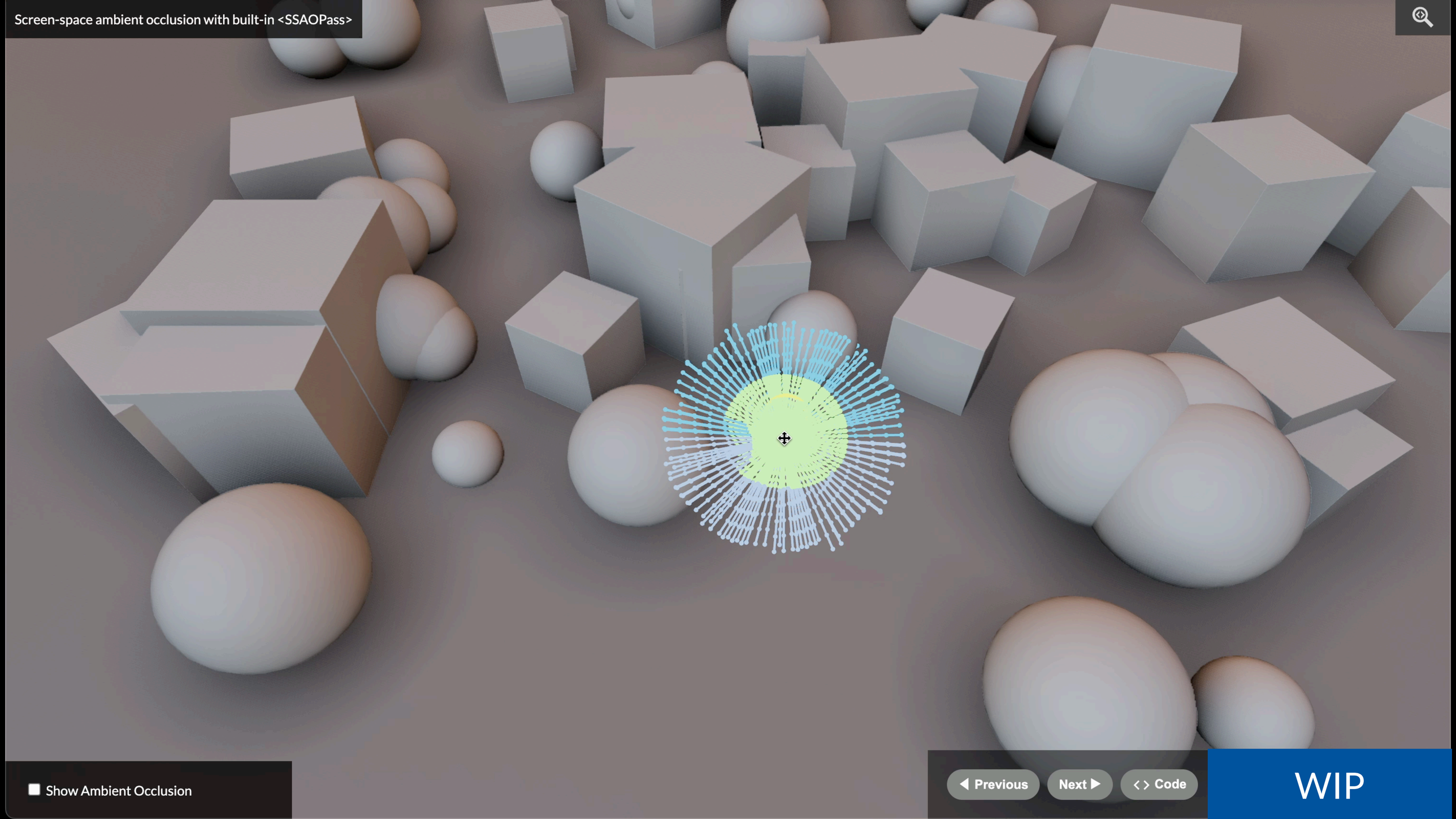
Screen-space ambient occlusion with built-in <SSAOPass>

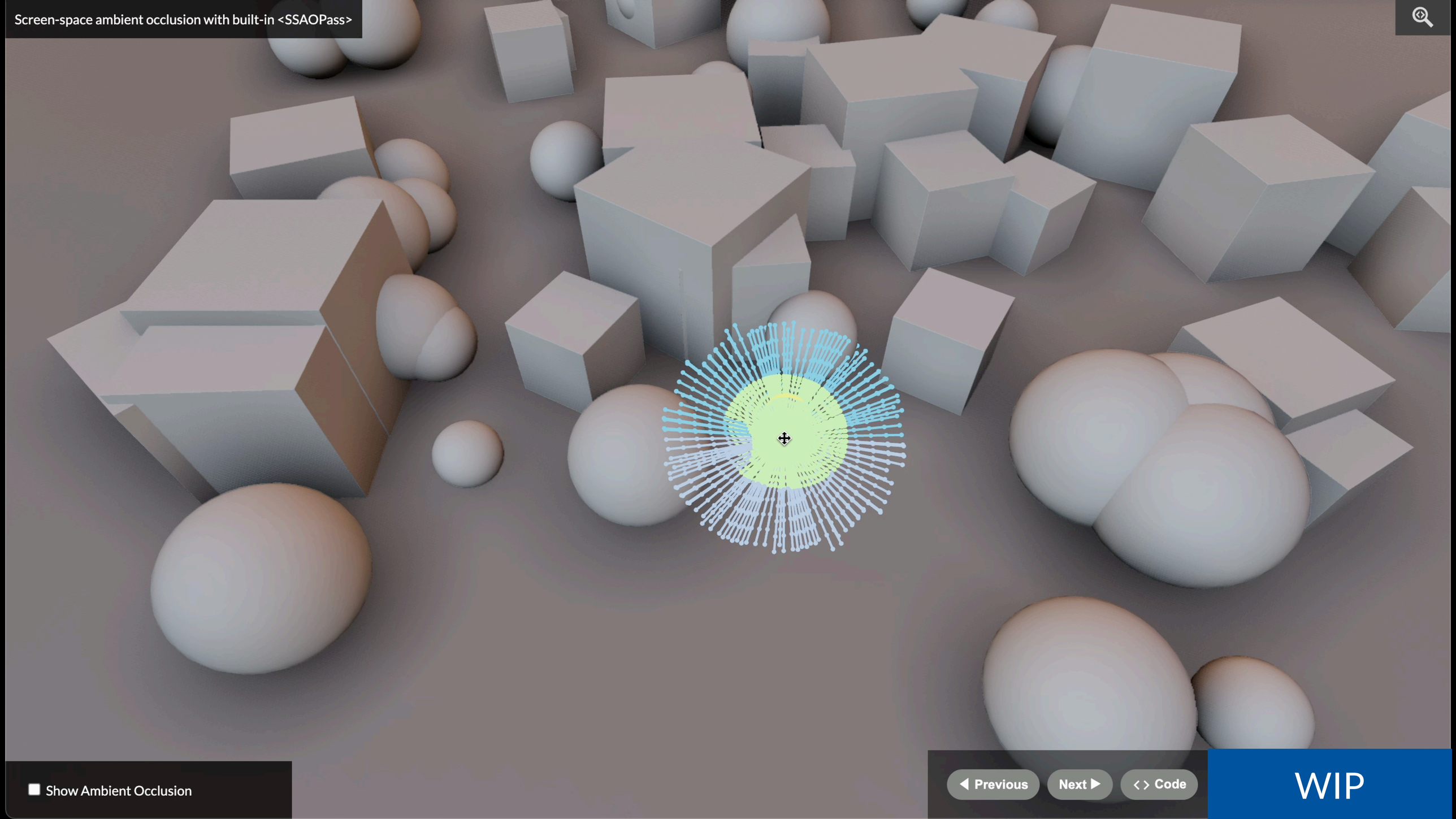


Show Ambient Occlusion

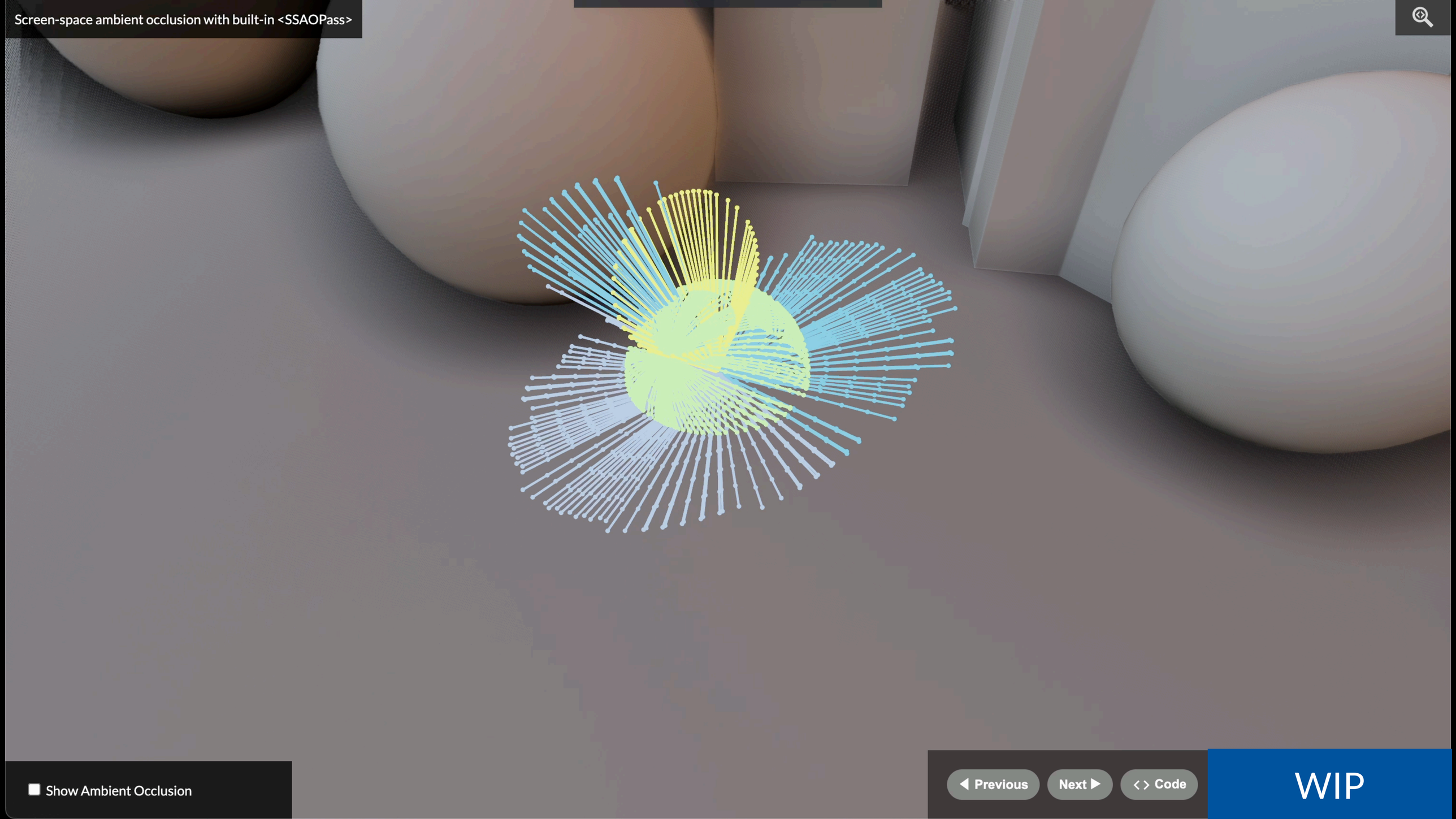
◀ Previous Next ▶ < > Code

WIP


Screen-space ambient occlusion with built-in <SSAOPass>
Show Ambient Occlusion
Previous
Next
Code
WIP


Screen-space ambient occlusion with built-in <SSAOPass>
Show Ambient Occlusion
Previous
Next
Code
WIP

Screen-space ambient occlusion with built-in <SSAOPass>

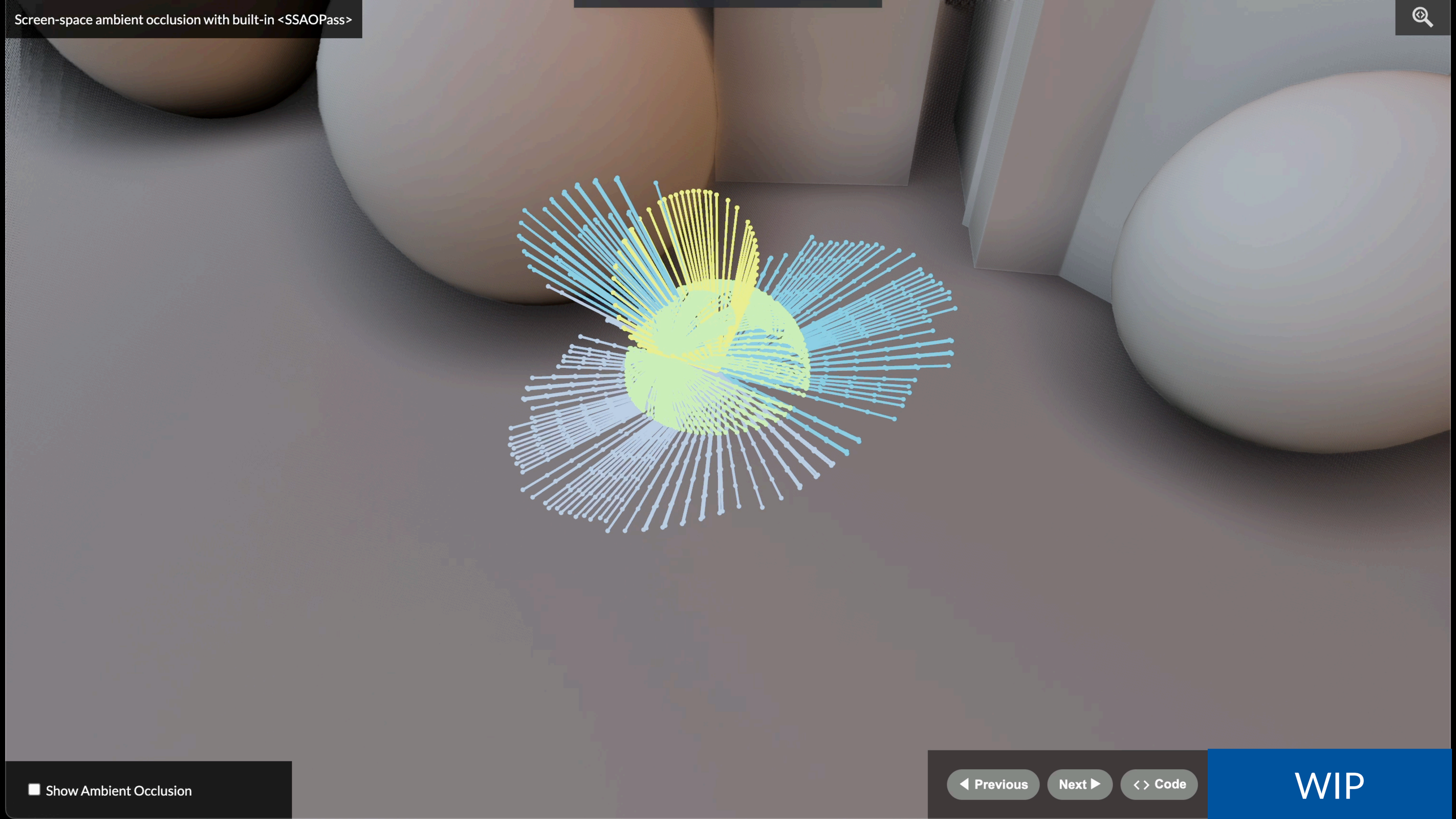


Show Ambient Occlusion

◀ Previous Next ▶ <> Code

WIP

Screen-space ambient occlusion with built-in <SSAOPass>



Show Ambient Occlusion

◀ Previous Next ▶ <> Code

WIP

Screen-space ambient occlusion with built-in <SSAOPass>

```
use '@use-gpu/wgsl/debug/print'::{ PrintData };

@link var<storage, read_write> data: PrintData;
@link var<storage, read_write> positions: array<vec4<f32>>;
@link var<storage, read_write> colors: array<vec4<f32>>;
@link var<storage, read_write> segments: array<i32>;

@export fn printPoint(position: vec4<f32>, color: vec4<f32>) {
  let index = atomicAdd(&data.vertex, 1u);
  positions[index] = position;
  colors[index] = color;
  segments[index] = 0;
}

@export fn printLine(start: vec4<f32>, end: vec4<f32>, color: vec4<f32>) {
  let index = atomicAdd(&data.vertex, 2u);
  positions[index] = start;
  positions[index + 1] = end;
  colors[index] = color;
  colors[index + 1] = color;
  segments[index] = 1;
  segments[index + 1] = 2;
}

@export fn printData(vector: vec4<f32>) {
  let index = atomicAdd(&data.vector, 1u);
  data.vectors[index] = vector;
}
```

Show Ambient Occlusion

◀ Previous Next ▶ < > Code

WIP

Screen-space ambient occlusion with built-in <SSAOPass>

```
use '@use-gpu/wgsl/debug/print'::{ PrintData };

@link var<storage, read_write> data: PrintData;
@link var<storage, read_write> positions: array<vec4<f32>>;
@link var<storage, read_write> colors: array<vec4<f32>>;
@link var<storage, read_write> segments: array<i32>;

@export fn printPoint(position: vec4<f32>, color: vec4<f32>) {
  let index = atomicAdd(&data.vertex, 1u);
  positions[index] = position;
  colors[index] = color;
  segments[index] = 0;
}

@export fn printLine(start: vec4<f32>, end: vec4<f32>, color: vec4<f32>) {
  let index = atomicAdd(&data.vertex, 2u);
  positions[index] = start;
  positions[index + 1] = end;
  colors[index] = color;
  colors[index + 1] = color;
  segments[index] = 1;
  segments[index + 1] = 2;
}

@export fn printData(vector: vec4<f32>) {
  let index = atomicAdd(&data.vector, 1u);
  data.vectors[index] = vector;
}
```

Show Ambient Occlusion

◀ Previous Next ▶ < > Code

WIP

Screen-space ambient occlusion with built-in <SSAOPass>

Imports

```wgsl
use '@use-gpu/wgsl/debug/print'::{ PrintData };

@link var<storage, read_write> data: PrintData;
@link var<storage, read_write> positions: array<vec4<f32>>;
@link var<storage, read_write> colors: array<vec4<f32>>;
@link var<storage, read_write> segments: array<i32>;

@export fn printPoint(position: vec4<f32>, color: vec4<f32>) {
  let index = atomicAdd(&data.vertex, 1u);
  positions[index] = position;
  colors[index] = color;
  segments[index] = 0;
}

@export fn printLine(start: vec4<f32>, end: vec4<f32>, color: vec4<f32>) {
  let index = atomicAdd(&data.vertex, 2u);
  positions[index] = start;
  positions[index + 1] = end;
  colors[index] = color;
  colors[index + 1] = color;
  segments[index] = 1;
  segments[index + 1] = 2;
}

@export fn printData(vector: vec4<f32>) {
  let index = atomicAdd(&data.vector, 1u);
  data.vectors[index] = vector;
}
```

Show Ambient Occlusion

◀ Previous Next ▶ < > Code

WIP

Screen-space ambient occlusion with built-in <SSAOPass>

Imports

Bindings

```wgsl
use '@use-gpu/wgsl/debug/print'::{ PrintData };

@link var<storage, read_write> data: PrintData;
@link var<storage, read_write> positions: array<vec4<f32>>;
@link var<storage, read_write> colors: array<vec4<f32>>;
@link var<storage, read_write> segments: array<i32>;

@export fn printPoint(position: vec4<f32>, color: vec4<f32>) {
  let index = atomicAdd(&data.vertex, 1u);
  positions[index] = position;
  colors[index] = color;
  segments[index] = 0;
}

@export fn printLine(start: vec4<f32>, end: vec4<f32>, color: vec4<f32>) {
  let index = atomicAdd(&data.vertex, 2u);
  positions[index] = start;
  positions[index + 1] = end;
  colors[index] = color;
  colors[index + 1] = color;
  segments[index] = 1;
  segments[index + 1] = 2;
}

@export fn printData(vector: vec4<f32>) {
  let index = atomicAdd(&data.vector, 1u);
  data.vectors[index] = vector;
}
```

☐ Show Ambient Occlusion

◀ Previous Next ▶ < > Code

WIP

Screen-space ambient occlusion with built-in <SSAOPass>

```wgsl
use '@use-gpu/wgsl/debug/print'::{ PrintData };

@link var<storage, read_write> data: PrintData;
@link var<storage, read_write> positions: array<vec4<f32>>;
@link var<storage, read_write> colors: array<vec4<f32>>;
@link var<storage, read_write> segments: array<i32>;

@export fn printPoint(position: vec4<f32>, color: vec4<f32>) {
  let index = atomicAdd(&data.vertex, 1u);
  positions[index] = position;
  colors[index] = color;
  segments[index] = 0;
}

@export fn printLine(start: vec4<f32>, end: vec4<f32>, color: vec4<f32>) {
  let index = atomicAdd(&data.vertex, 2u);
  positions[index] = start;
  positions[index + 1] = end;
  colors[index] = color;
  colors[index + 1] = color;
  segments[index] = 1;
  segments[index + 1] = 2;
}

@export fn printData(vector: vec4<f32>) {
  let index = atomicAdd(&data.vector, 1u);
  data.vectors[index] = vector;
}
```

Imports

Bindings

Print

Show Ambient Occlusion

◀ Previous Next ▶ <> Code

WIP

Screen-space ambient occlusion with built-in <SSAOPass>

```tsx
export const DebugLineHelper: LC<DebugLineHelperProps> = (props: DebugLineHelperProps) => {
  const {
    count = 1024,
    render,
  } = props;

  const atomicArray   = useOne(() => new Uint32Array(16));
  const atomicStorage = useRawSource(atomicArray, 'u32', READ_WRITE_SOURCE);

  const [debugPositions, allocatePositions] = useScratchSource('vec4<f32>', READ_WRITE_SOURCE);
  const [debugColors,    allocateColors]    = useScratchSource('vec4<f32>', READ_WRITE_SOURCE);
  const [debugSegments,  allocateSegments]  = useScratchSource('i32', READ_WRITE_SOURCE);
  allocatePositions(count);
  allocateColors(count);
  allocateSegments(count);

  const device = useDeviceContext();

  const helper = useMemo(() => {
    const swap = () => {
      clearBuffer(device, debugPositions.buffer);
      clearBuffer(device, debugColors.buffer);
      clearBuffer(device, debugSegments.buffer);
      clearBuffer(device, atomicStorage.buffer);
    }

    const attributes = {
      counter: getDerivedSource(atomicStorage, { readWrite: false, format: 'u32' }),
      positions: getDerivedSource(debugPositions, { readWrite: false }),
```

Show Ambient Occlusion

◀ Previous | Next ▶ | <> Code

WIP

Screen-space ambient occlusion with built-in <SSAOPass>

```tsx
export const DebugLineHelper: LC<DebugLineHelperProps> = (props: DebugLineHelperProps) => {
  const {
    count = 1024,
    render,
  } = props;

  const atomicArray   = useOne(() => new Uint32Array(16));
  const atomicStorage = useRawSource(atomicArray, 'u32', READ_WRITE_SOURCE);

  const [debugPositions, allocatePositions] = useScratchSource('vec4<f32>', READ_WRITE_SOURCE);
  const [debugColors,    allocateColors]    = useScratchSource('vec4<f32>', READ_WRITE_SOURCE);
  const [debugSegments,  allocateSegments]  = useScratchSource('i32', READ_WRITE_SOURCE);
  allocatePositions(count);
  allocateColors(count);
  allocateSegments(count);

  const device = useDeviceContext();

  const helper = useMemo(() => {
    const swap = () => {
      clearBuffer(device, debugPositions.buffer);
      clearBuffer(device, debugColors.buffer);
      clearBuffer(device, debugSegments.buffer);
      clearBuffer(device, atomicStorage.buffer);
    }

    const attributes = {
      counter: getDerivedSource(atomicStorage, { readWrite: false, format: 'u32' }),
      positions: getDerivedSource(debugPositions, { readWrite: false }),
```

Show Ambient Occlusion

◀ Previous | Next ▶ | <> Code

WIP

Screen-space ambient occlusion with built-in <SSAOPass>

```tsx
export const DebugLineHelper: LC<DebugLineHelperProps> = (props: DebugLineHelperProps) => {
  const {
    count = 1024,
    render,
  } = props;

  const atomicArray   = useOne(() => new Uint32Array(16));
  const atomicStorage = useRawSource(atomicArray, 'u32', READ_WRITE_SOURCE);

  const [debugPositions, allocatePositions] = useScratchSource('vec4<f32>', READ_WRITE_SOURCE);
  const [debugColors,    allocateColors]    = useScratchSource('vec4<f32>', READ_WRITE_SOURCE);
  const [debugSegments,  allocateSegments]  = useScratchSource('i32', READ_WRITE_SOURCE);
  allocatePositions(count);
  allocateColors(count);
  allocateSegments(count);

  const device = useDeviceContext();

  const helper = useMemo(() => {
    const swap = () => {
      clearBuffer(device, debugPositions.buffer);
      clearBuffer(device, debugColors.buffer);
      clearBuffer(device, debugSegments.buffer);
      clearBuffer(device, atomicStorage.buffer);
    }

    const attributes = {
      counter: getDerivedSource(atomicStorage, { readWrite: false, format: 'u32' }),
      positions: getDerivedSource(debugPositions, { readWrite: false }),
```

Live component

Show Ambient Occlusion

◀ Previous Next ▶ <> Code

WIP

Screen-space ambient occlusion with built-in <SSAOPass>

```tsx
export const DebugLineHelper: LC<DebugLineHelperProps> = (props: DebugLineHelperProps) => {
  const {
    count = 1024,
    render,
  } = props;

  const atomicArray   = useOne(() => new Uint32Array(16));
  const atomicStorage = useRawSource(atomicArray, 'u32', READ_WRITE_SOURCE);

  const [debugPositions, allocatePositions] = useScratchSource('vec4<f32>', READ_WRITE_SOURCE);
  const [debugColors,    allocateColors]    = useScratchSource('vec4<f32>', READ_WRITE_SOURCE);
  const [debugSegments,  allocateSegments]  = useScratchSource('i32', READ_WRITE_SOURCE);
  allocatePositions(count);
  allocateColors(count);
  allocateSegments(count);

  const device = useDeviceContext();

  const helper = useMemo(() => {
    const swap = () => {
      clearBuffer(device, debugPositions.buffer);
      clearBuffer(device, debugColors.buffer);
      clearBuffer(device, debugSegments.buffer);
      clearBuffer(device, atomicStorage.buffer);
    }

    const attributes = {
      counter: getDerivedSource(atomicStorage, { readWrite: false, format: 'u32' }),
      positions: getDerivedSource(debugPositions, { readWrite: false }),
```

Live component

Synced array

Show Ambient Occlusion

◀ Previous Next ▶ <> Code

WIP

Screen-space ambient occlusion with built-in <SSAOPass>

```ts
export const DebugLineHelper: LC<DebugLineHelperProps> = (props: DebugLineHelperProps) => {
  const {
    count = 1024,
    render,
  } = props;

  const atomicArray   = useOne(() => new Uint32Array(16));
  const atomicStorage = useRawSource(atomicArray, 'u32', READ_WRITE_SOURCE);

  const [debugPositions, allocatePositions] = useScratchSource('vec4<f32>', READ_WRITE_SOURCE);
  const [debugColors,    allocateColors]    = useScratchSource('vec4<f32>', READ_WRITE_SOURCE);
  const [debugSegments,  allocateSegments]  = useScratchSource('i32', READ_WRITE_SOURCE);
  allocatePositions(count);
  allocateColors(count);
  allocateSegments(count);

  const device = useDeviceContext();

  const helper = useMemo(() => {
    const swap = () => {
      clearBuffer(device, debugPositions.buffer);
      clearBuffer(device, debugColors.buffer);
      clearBuffer(device, debugSegments.buffer);
      clearBuffer(device, atomicStorage.buffer);
    }

    const attributes = {
      counter: getDerivedSource(atomicStorage, { readWrite: false, format: 'u32' }),
      positions: getDerivedSource(debugPositions, { readWrite: false }),
```

Live component

Synced array

GPU-only array

WIP

Show Ambient Occlusion

◀ Previous Next ▶ <> Code

Screen-space ambient occlusion with built-in <SSAOPass>

```
60      const attributes = {
61        counter: getDerivedSource(atomicStorage, { readWrite: false, format: 'u32' }),
62        positions: getDerivedSource(debugPositions, { readWrite: false }),
63        colors: getDerivedSource(debugColors, { readWrite: false }),
64        segments: getDerivedSource(debugSegments, { readWrite: false }),
65      };
66
67      const target = {
68        counter: atomicStorage,
69        positions: debugPositions,
70        colors: debugColors,
71        segments: debugSegments,
72      };
73
74      const boundEmitter = getShader(debugWGSL, [target.counter, target.positions, target.colors, target.segments]);
75      const emitPoint = bindEntryPoint(boundEmitter, 'emitPoint');
76      const emitLine = bindEntryPoint(boundEmitter, 'emitLine');
77      const shaders = {emitPoint, emitLine};
```

Show Ambient Occlusion

◀ Previous Next ▶ <> Code

WIP

Screen-space ambient occlusion with built-in <SSAOPass>

```
60      const attributes = {
61        counter: getDerivedSource(atomicStorage, { readWrite: false, format: 'u32' }),
62        positions: getDerivedSource(debugPositions, { readWrite: false }),
63        colors: getDerivedSource(debugColors, { readWrite: false }),
64        segments: getDerivedSource(debugSegments, { readWrite: false }),
65      };
66
67      const target = {
68        counter: atomicStorage,
69        positions: debugPositions,
70        colors: debugColors,
71        segments: debugSegments,
72      };
73
74      const boundEmitter = getShader(debugWGSL, [target.counter, target.positions, target.colors, target.segments]);
75      const emitPoint = bindEntryPoint(boundEmitter, 'emitPoint');
76      const emitLine = bindEntryPoint(boundEmitter, 'emitLine');
77      const shaders = {emitPoint, emitLine};
```

Show Ambient Occlusion

◀ Previous Next ▶ <> Code

WIP

Screen-space ambient occlusion with built-in <SSAOPass>

```
60    const attributes = {
61      counter: getDerivedSource(atomicStorage, { readWrite: false, format: 'u32' }),
62      positions: getDerivedSource(debugPositions, { readWrite: false }),
63      colors: getDerivedSource(debugColors, { readWrite: false }),
64      segments: getDerivedSource(debugSegments, { readWrite: false }),
65    };
66
67    const target = {
68      counter: atomicStorage,
69      positions: debugPositions,
70      colors: debugColors,
71      segments: debugSegments,
72    };
73
74    const boundEmitter = getShader(debugWGSL, [target.counter, target.positions, target.colors, target.segments]);
75    const emitPoint = bindEntryPoint(boundEmitter, 'emitPoint');
76    const emitLine = bindEntryPoint(boundEmitter, 'emitLine');
77    const shaders = {emitPoint, emitLine};
```

Bind shader to storage

Show Ambient Occlusion

◀ Previous Next ▶ <> Code

WIP

```
130
131     var s1 = to3D(clipToView(vec4<f32>(clip1, d1, 1.0)));
132     var s2 = to3D(clipToView(vec4<f32>(clip2, d2, 1.0)));
133
134     if (HAS_DEBUG_PICKING && log) {
135       printLine(viewToWorld(vec4<f32>(position, 1.0)), viewToWorld(vec4<f32>(s1, 1.0)), vec4<f32>(0.3, 0.75, 1.0, 1.0));
136       printLine(viewToWorld(vec4<f32>(position, 1.0)), viewToWorld(vec4<f32>(s2, 1.0)), vec4<f32>(0.6, 0.75, 1.0, 1.0));
137     }
138
139     let os1 = normalize(projectOnDirection(da, s1 - position));
```

```
183
184   if (HAS_DEBUG_PICKING && log) {
185     let angle1 = th1 - gamma;
186     let angle2 = th2 - gamma;
187     for (var i = 0; i < 16; i++) {
188       let pos1 = position + radius * 0.5 * slerpAngle(upProj, side, mix(angle1, angle2, f32(i)/16.0));
189       let pos2 = position + radius * 0.5 * slerpAngle(upProj, side, mix(angle1, angle2, f32(i+1)/16.0));
190       printLine(
191         viewToWorld(vec4<f32>(pos1, 1.0)),
192         viewToWorld(vec4<f32>(pos2, 1.0)),
193         vec4<f32>(0.5, 1.2, 0.5, 1.0)
194       );
195     }
196
197     printLine(
198       viewToWorld(vec4<f32>(position, 1.0)),
199       viewToWorld(vec4<f32>(position + nproj, 1.0)),
200       vec4<f32>(1.0, 1.2, 0.0, 1.0)
201     );
202
```

Show Ambient Occlusion

```
130
131     var s1 = to3D(clipToView(vec4<f32>(clip1, d1, 1.0)));
132     var s2 = to3D(clipToView(vec4<f32>(clip2, d2, 1.0)));
133
134     if (HAS_DEBUG_PICKING && log) {
135       printLine(viewToWorld(vec4<f32>(position, 1.0)), viewToWorld(vec4<f32>(s1, 1.0)), vec4<f32>(0.3, 0.75, 1.0, 1.0));
136       printLine(viewToWorld(vec4<f32>(position, 1.0)), viewToWorld(vec4<f32>(s2, 1.0)), vec4<f32>(0.6, 0.75, 1.0, 1.0));
137     }
138
139     let os1 = normalize(projectOnDirection(da, s1 - position));
```

```
183
184   if (HAS_DEBUG_PICKING && log) {
185     let angle1 = th1 - gamma;
186     let angle2 = th2 - gamma;
187     for (var i = 0; i < 16; i++) {
188       let pos1 = position + radius * 0.5 * slerpAngle(upProj, side, mix(angle1, angle2, f32(i)/16.0));
189       let pos2 = position + radius * 0.5 * slerpAngle(upProj, side, mix(angle1, angle2, f32(i+1)/16.0));
190       printLine(
191         viewToWorld(vec4<f32>(pos1, 1.0)),
192         viewToWorld(vec4<f32>(pos2, 1.0)),
193         vec4<f32>(0.5, 1.2, 0.5, 1.0)
194       );
195     }
196
197     printLine(
198       viewToWorld(vec4<f32>(position, 1.0)),
199       viewToWorld(vec4<f32>(position + nproj, 1.0)),
200       vec4<f32>(1.0, 1.2, 0.0, 1.0)
201     );
202
```

Show Ambient Occlusion

```
130
131     var s1 = to3D(clipToView(vec4<f32>(clip1, d1, 1.0)));
132     var s2 = to3D(clipToView(vec4<f32>(clip2, d2, 1.0)));
133
134     if (HAS_DEBUG_PICKING && log) {
135       printLine(viewToWorld(vec4<f32>(position, 1.0)), viewToWorld(vec4<f32>(s1, 1.0)), vec4<f32>(0.3, 0.75, 1.0, 1.0));
136       printLine(viewToWorld(vec4<f32>(position, 1.0)), viewToWorld(vec4<f32>(s2, 1.0)), vec4<f32>(0.6, 0.75, 1.0, 1.0));
137
138
139                                                    la, s1 - position));
```

Shader: Print if pixel is being **inspected/logged**

```
183
184   if (HAS_DEBUG_PICKING && log) {
185     let angle1 = th1 - gamma;
186     let angle2 = th2 - gamma;
187     for (var i = 0; i < 16; i++) {
188       let pos1 = position + radius * 0.5 * slerpAngle(upProj, side, mix(angle1, angle2, f32(i)/16.0));
189       let pos2 = position + radius * 0.5 * slerpAngle(upProj, side, mix(angle1, angle2, f32(i+1)/16.0));
190       printLine(
191         viewToWorld(vec4<f32>(pos1, 1.0)),
192         viewToWorld(vec4<f32>(pos2, 1.0)),
193         vec4<f32>(0.5, 1.2, 0.5, 1.0)
194       );
195     }
196
197     printLine(
198       viewToWorld(vec4<f32>(position, 1.0)),
199       viewToWorld(vec4<f32>(position + nproj, 1.0)),
200       vec4<f32>(1.0, 1.2, 0.0, 1.0)
201     );
202
```

Memo(Pass)
Memo(FullScreenRenderer)
Reconcile
Memo(RawFullScreen)
Root(PassReconciler)
Fiber
Resume(MultiGather)
Memo(ColorPass)
HTML
Memo(Routes)
PagePicker
HTML
FPSToggle
Resume(CursorState)
Root(QueueReconciler)
Resume(Gather)
oot(QueueReconciler)
Resume(Gather)
TML

Mounted
Updated
Rendered By
Dependency
Portal
Yeet
Quote
Highlight
Output
Layout
React

Double click to focus a sub-tree

Load a .glb model using the GLTF package. Supports PBR materials.

Props Fiber Vertex **Fragment** Geometry

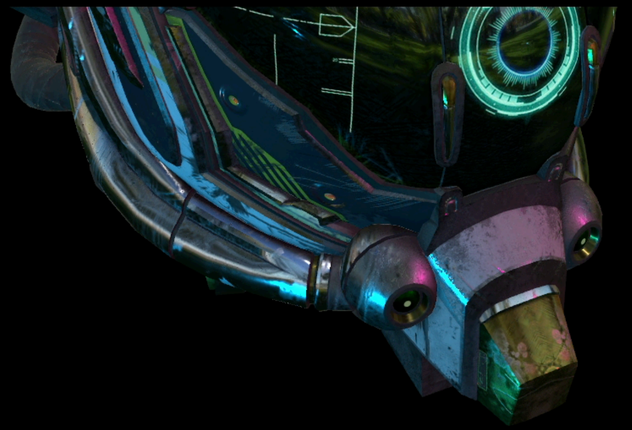
Constants
› **_VT_1_getGain** {uniform: {...}, constant: {...}}

Bindings
⌄ **getTexture** Object
› **uniform** {name: getTexture, format: vec4<f32>, args: [vec2<f32>], attr: [optional, link]}...
⌄ **texture** Object
› **texture** GPUTexture{}
› **view** GPUTextureView{}
› **sampler** {}
layout texture_2d<f32>
format rgba16float
variant textureSample
absolute false
colorSpace linear
› **size** [2200, 1714]
volatile 0
version 383
swap swap(...){if(!d)return;const{current:e}=Q,t=M.len...
history undefined

Shader (ds4xmibljl-a4×6bhxtkv-1)

Hot Reload ⌘ + S

```
1  const IS_OPAQUE = true;
```



Environment Map
Park
Approximate SH

◀ Previous Next ▶ <> Code Geometry - GLTF

Memo(Pass)
Memo(FullScreenRenderer)
Reconcile
Memo(RawFullScreen)
Root(PassReconciler)
Fiber
Resume(MultiGather)
Memo(ColorPass)
HTML
Memo(Routes)
PagePicker
HTML
FPSToggle
Resume(CursorState)
Root(QueueReconciler)
Resume(Gather)
oot(QueueReconciler)
Resume(Gather)
TML

Mounted
Updated
Rendered By
Dependency
Portal
Yeet
Quote
Highlight
Output
Layout
React

Double click to focus a sub-tree

Load a .glb model using the GLTF package. Supports PBR materials.

Props Fiber Vertex **Fragment** Geometry

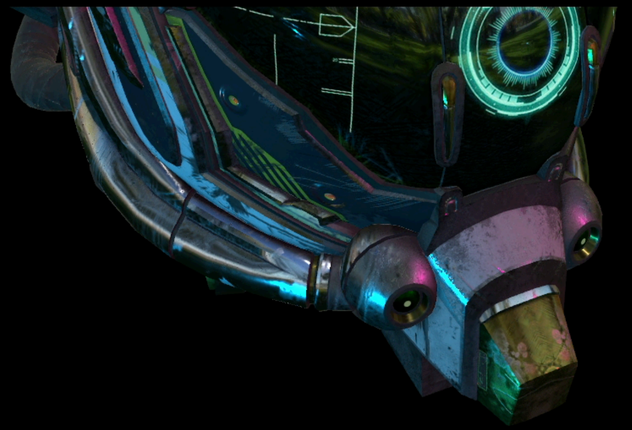
Constants
› **_VT_1_getGain** {uniform: {...}, constant: {...}}

Bindings
⌄ **getTexture** Object
› **uniform** {name: getTexture, format: vec4<f32>, args: [vec2<f32>], attr: [optional, link]}...
⌄ **texture** Object
› **texture** GPUTexture{}
› **view** GPUTextureView{}
› **sampler** {}
layout texture_2d<f32>
format rgba16float
variant textureSample
absolute false
colorSpace linear
› **size** [2200, 1714]
volatile 0
version 383
swap swap(...){if(!d)return;const{current:e}=Q,t=M.len...
history undefined

Shader (ds4xmibljl-a4×6bhxtkv-1)

Hot Reload ⌘ + S

```
1  const IS_OPAQUE = true;
```



Environment Map
Park
Approximate SH

◀ Previous Next ▶ <> Code Geometry - GLTF

Memo(Pass)
Memo(FullScreenRenderer)
Reconcile
Memo(RawFullScreen)
Root(PassReconciler)
Fiber
Resume(MultiGather)
Memo(ColorPass)
HTML
Memo(Routes)
PagePicker
HTML
FPSToggle
Resume(CursorState)
Root(QueueReconciler)
Resume(Gather)
oot(QueueReconciler)
Resume(Gather)
TML

Mounted
Updated
Rendered By
Dependency
Portal
Yeet
Quote
Highlight
Output
Layout
React

Double click to focus a sub-tree

Load a .glb model using the GLTF package. Supports PBR materials.

Props Fiber Vertex **Fragment** Geometry

Constants
› **_VT_1_getGain** {uniform: {...}, constant: {...}}

Bindings
⌄ **getTexture** Object
› **uniform** {name: getTexture, format: vec4<f32>, args: [vec2<f32>], attr: [optional, link]}...
⌄ **texture** Object
› **texture** GPUTexture{}
› **view** GPUTextureView{}
› **sampler** {}
layout texture_2d<f32>
format rgba16float
variant textureSample
absolute false
colorSpace linear
› **size** [2200, 1714]
volatile 0
version 383
swap swap(...){if(!d)return;const{current:e}=Q,t=M.len...
history undefined

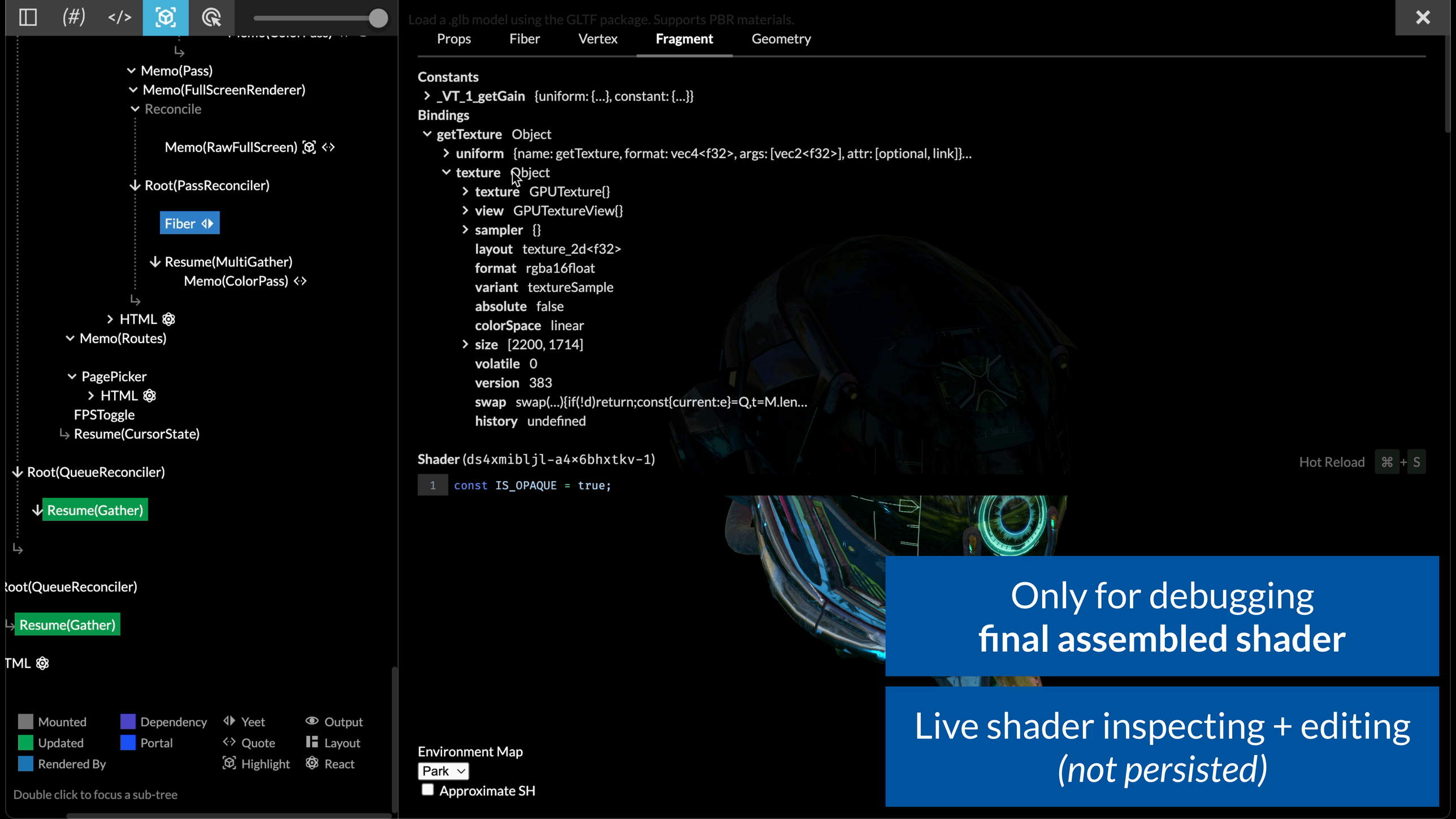
Shader (ds4xmibljl-a4×6bhxtkv-1)

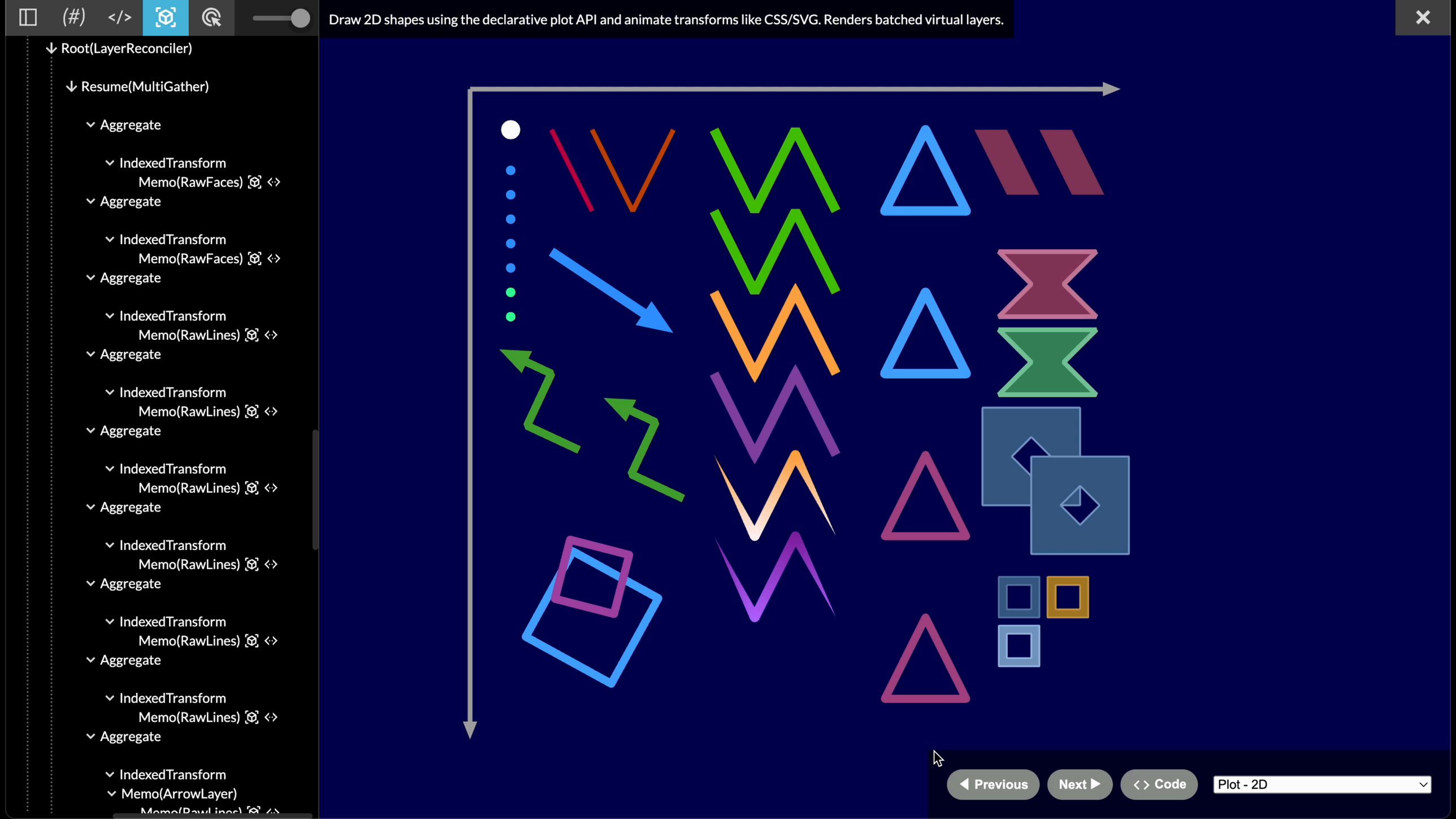
```
1  const IS_OPAQUE = true;
```

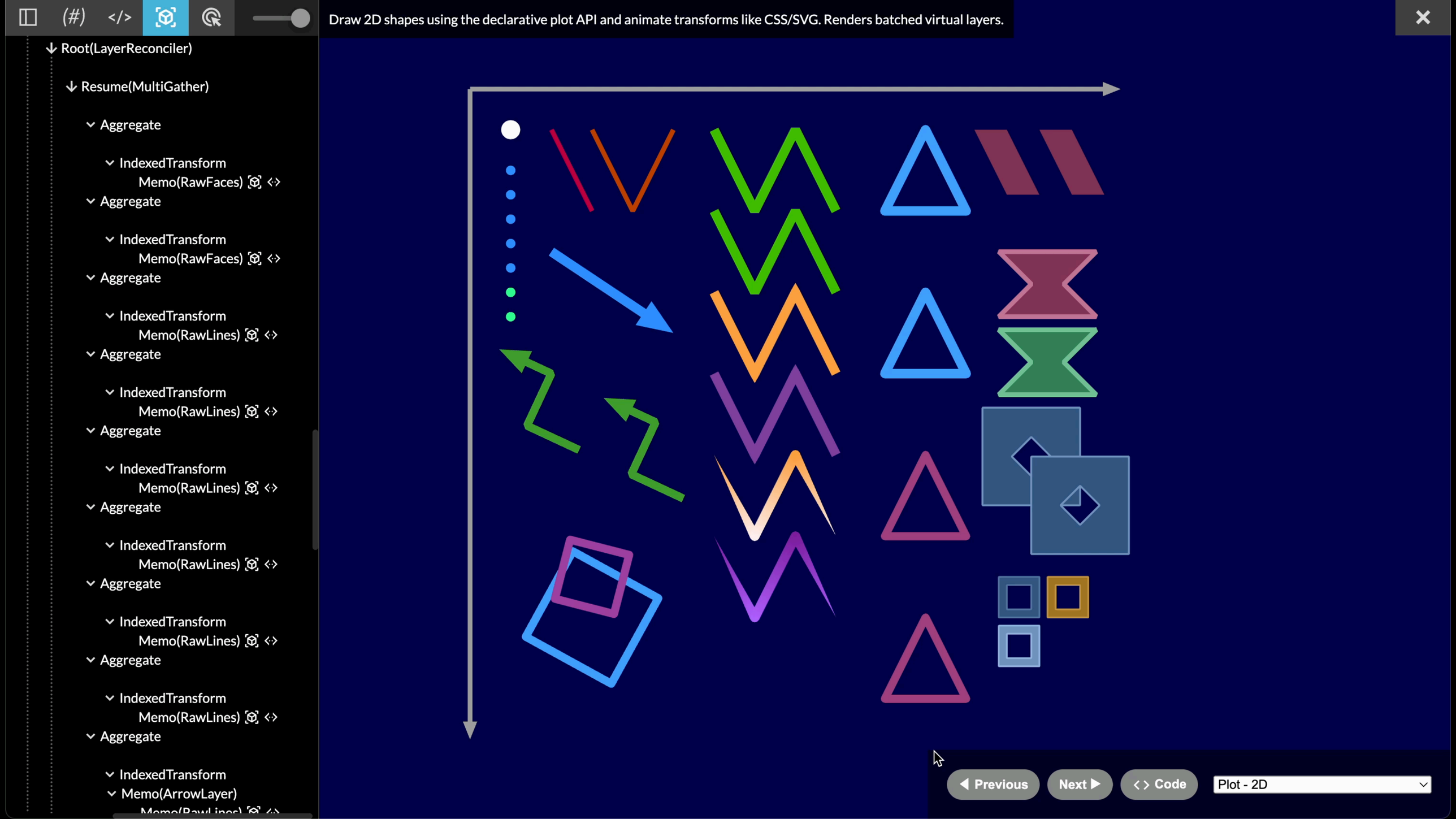
Hot Reload ⌘ + S

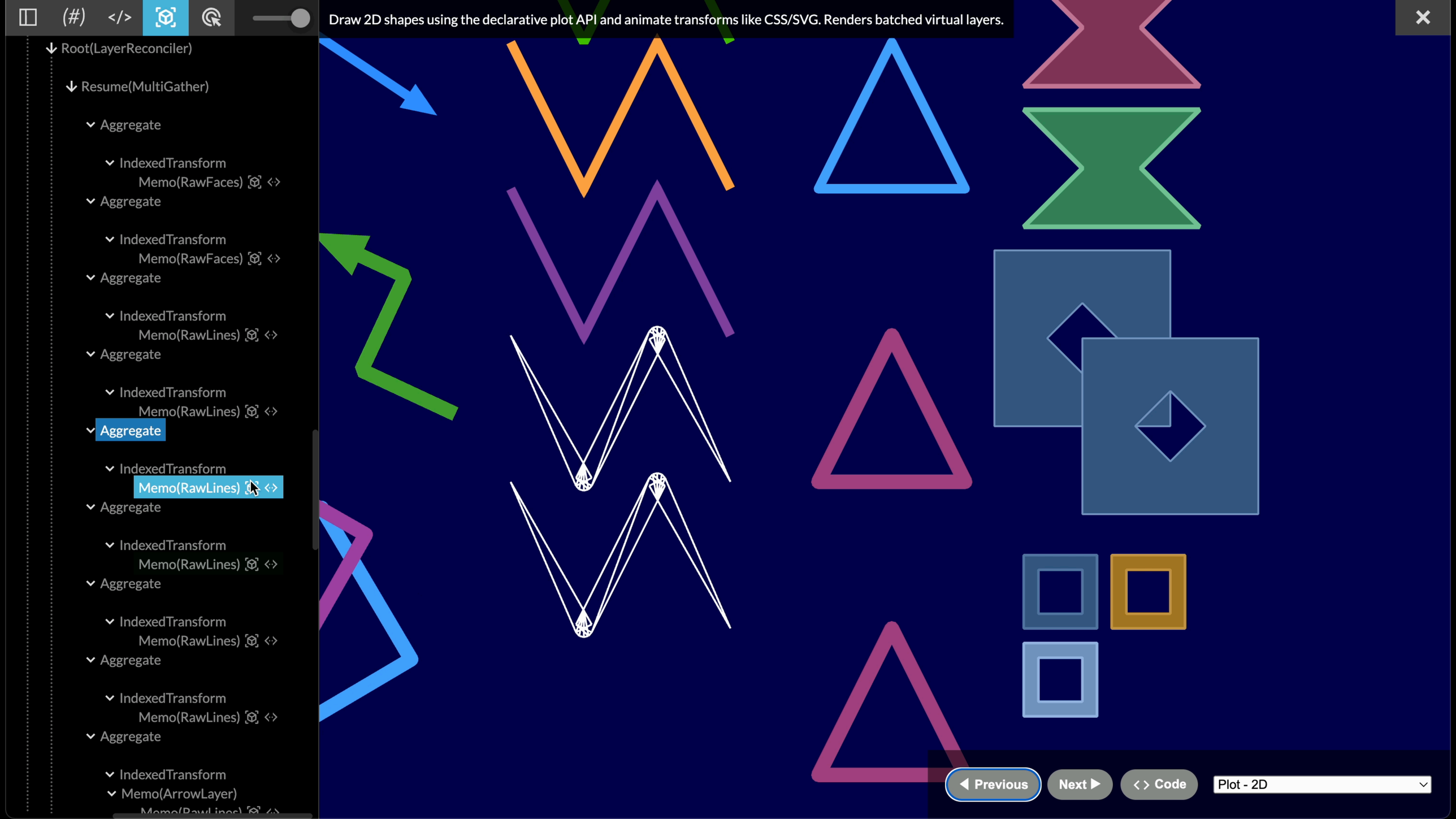
Environment Map
Park
Approximate SH

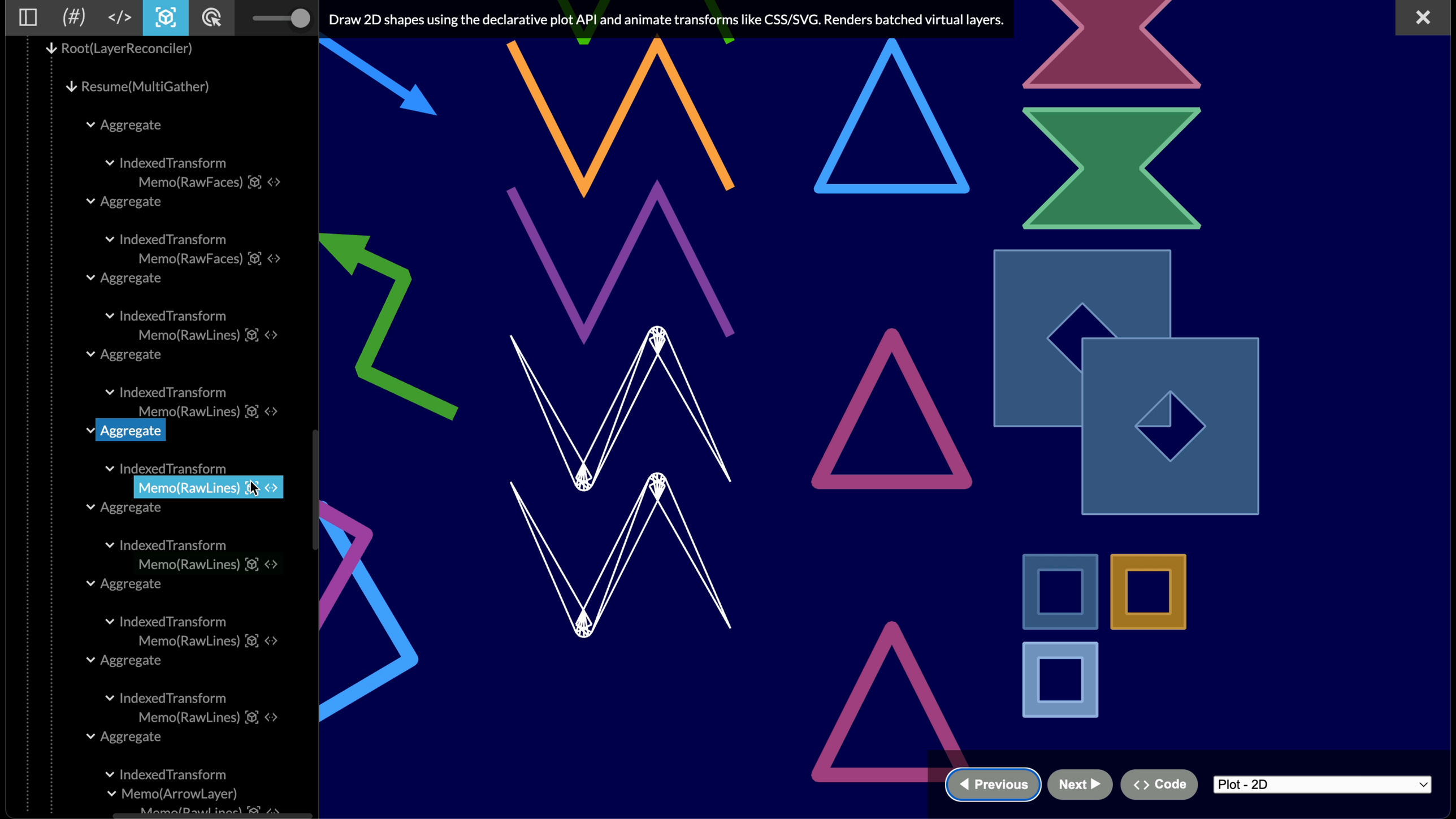
Live shader inspecting + editing
(not persisted)

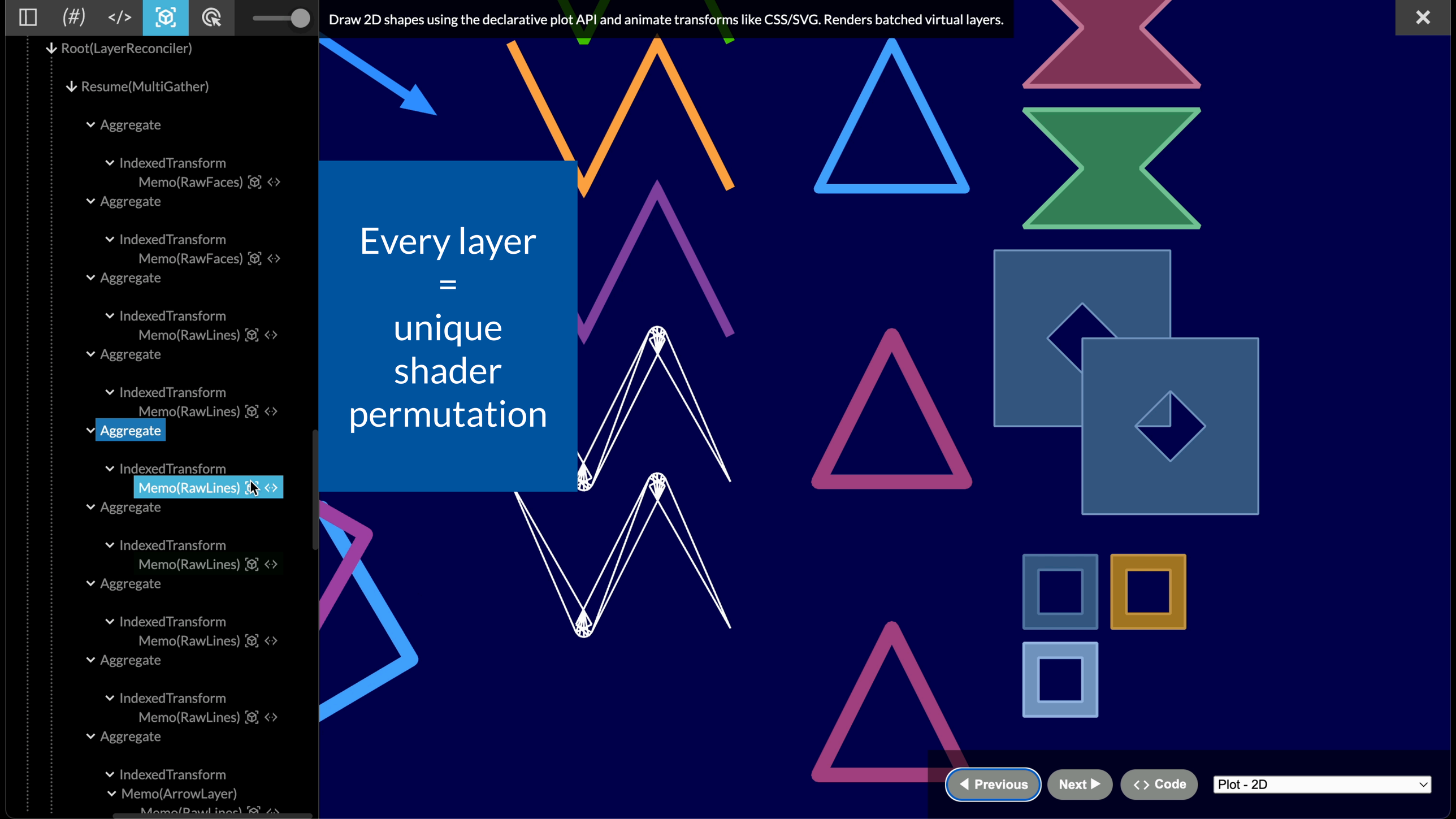

Memo(Pass)
Memo(FullScreenRenderer)
Reconcile
Memo(RawFullScreen)
Root(PassReconciler)
Fiber
Resume(MultiGather)
Memo(ColorPass)
HTML
Memo(Routes)
PagePicker
HTML
FPSToggle
Resume(CursorState)
Root(QueueReconciler)
Resume(Gather)
Resume(Gather)
Mounted
Updated
Rendered By
Dependency
Portal
Yeet
Quote
Highlight
Output
Layout
React
Double click to focus a sub-tree
Load a .glb model using the GLTF package. Supports PBR materials.
Props
Fiber
Vertex
Fragment
Geometry
Constants
_VT_1_getGain {uniform: {...}, constant: {...}}
Bindings
getTexture Object
uniform {name: getTexture, format: vec4<f32>, args: [vec2<f32>], attr: [optional, link]}...
texture Object
texture GPUTexture{}
view GPUTextureView{}
sampler {}
layout texture_2d<f32>
format rgba16float
variant textureSample
absolute false
colorSpace linear
size [2200, 1714]
volatile 0
version 383
swap swap(...){if(!d)return;const{current:e}=Q,t=M.len...
history undefined
Shader (ds4xmibljl-a4×6bhxtkv-1)
1 const IS_OPAQUE = true;
Hot Reload ⌘ + S
Environment Map
Park
Approximate SH
Only for debugging
final assembled shader
Live shader inspecting + editing
(not persisted)

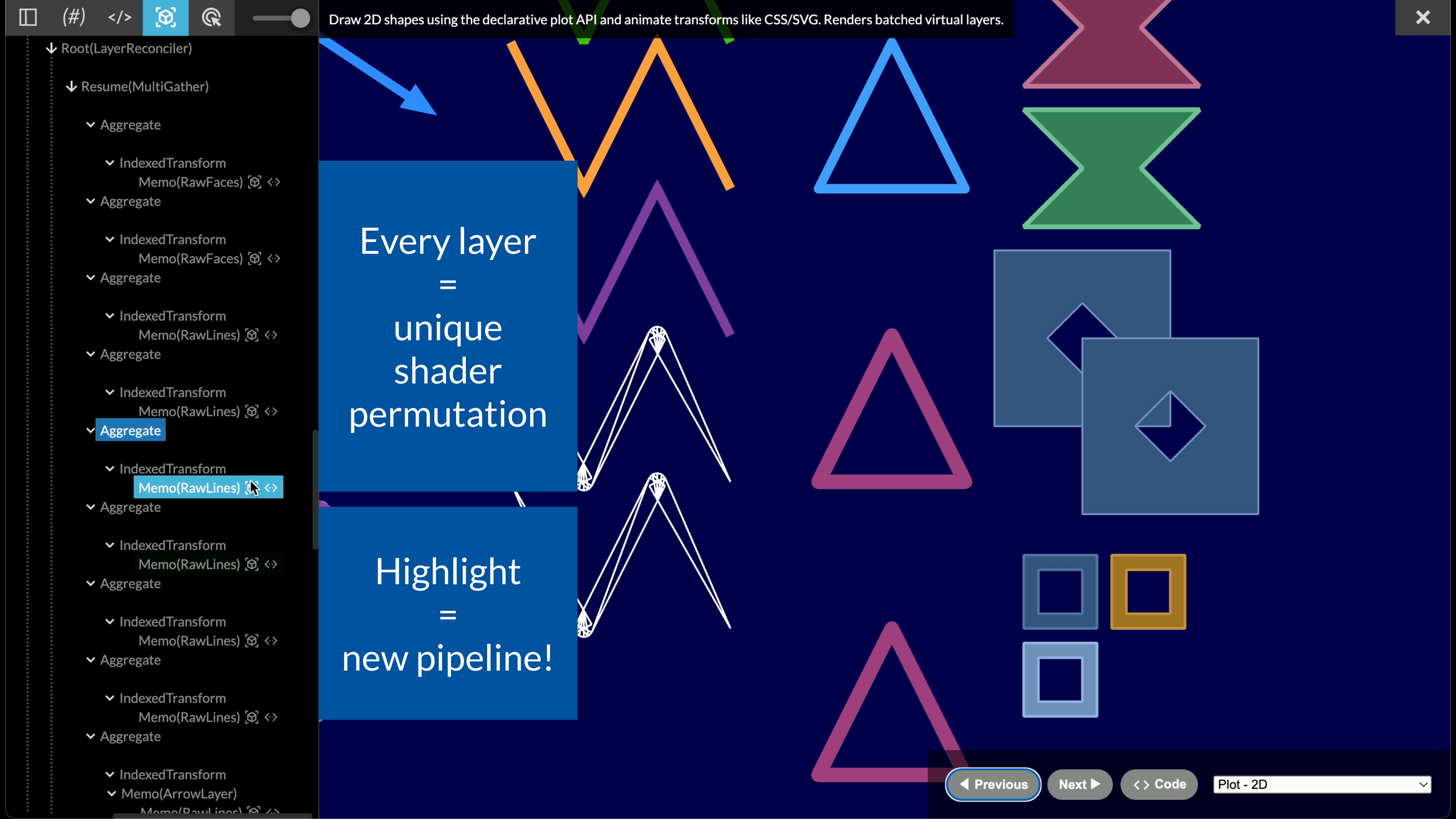

Draw 2D shapes using the declarative plot API and animate transforms like CSS/SVG. Renders batched virtual layers.
Root(LayerReconciler)
Resume(MultiGather)
Aggregate
IndexedTransform
Memo(RawFaces)
Aggregate
IndexedTransform
Memo(RawFaces)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(ArrowLayer)
Previous
Next
Code
Plot - 2D

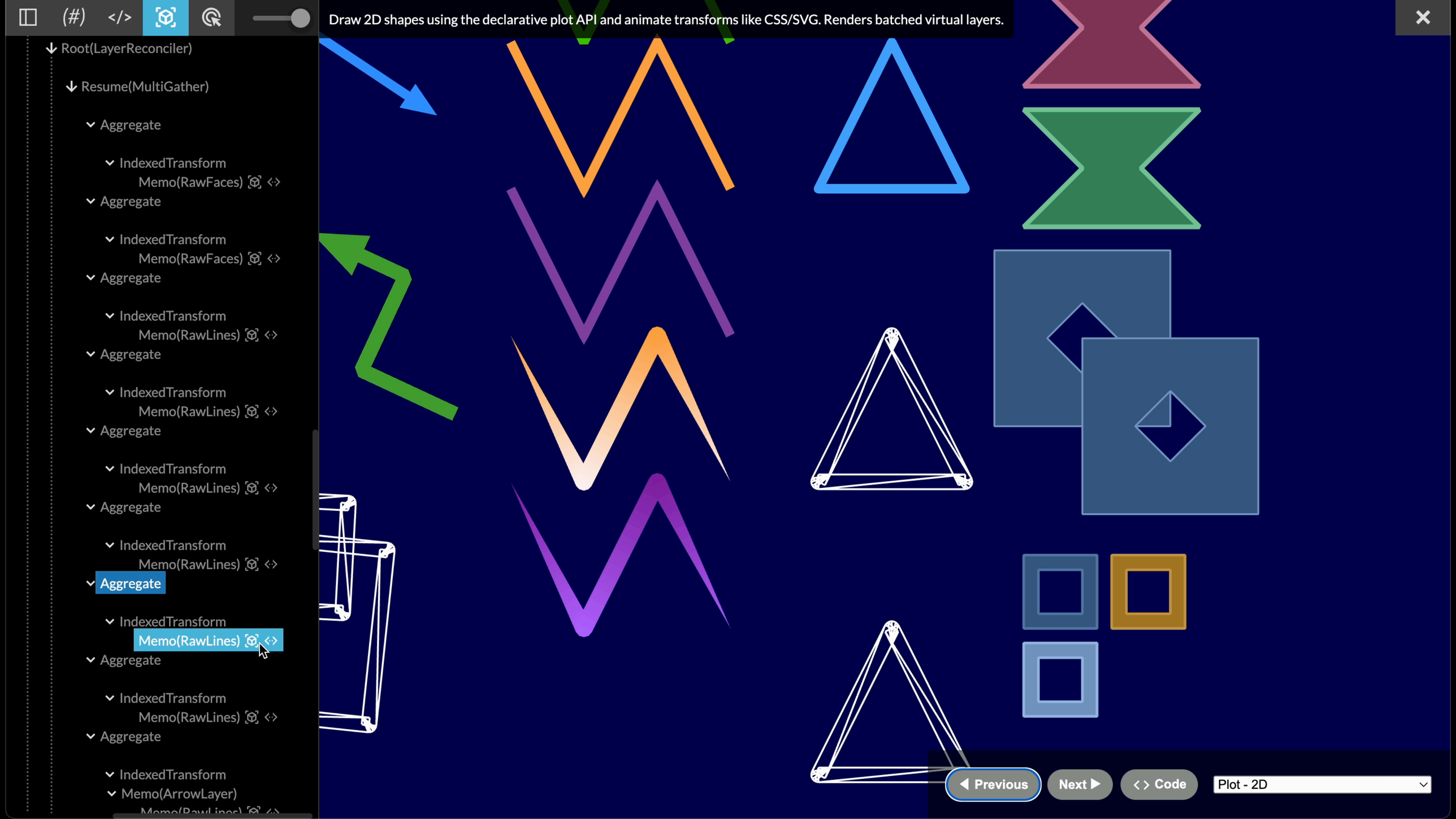

Draw 2D shapes using the declarative plot API and animate transforms like CSS/SVG. Renders batched virtual layers.
Root(LayerReconciler)
Resume(MultiGather)
Aggregate
IndexedTransform
Memo(RawFaces)
Aggregate
IndexedTransform
Memo(RawFaces)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(ArrowLayer)
Previous
Next
Code
Plot - 2D

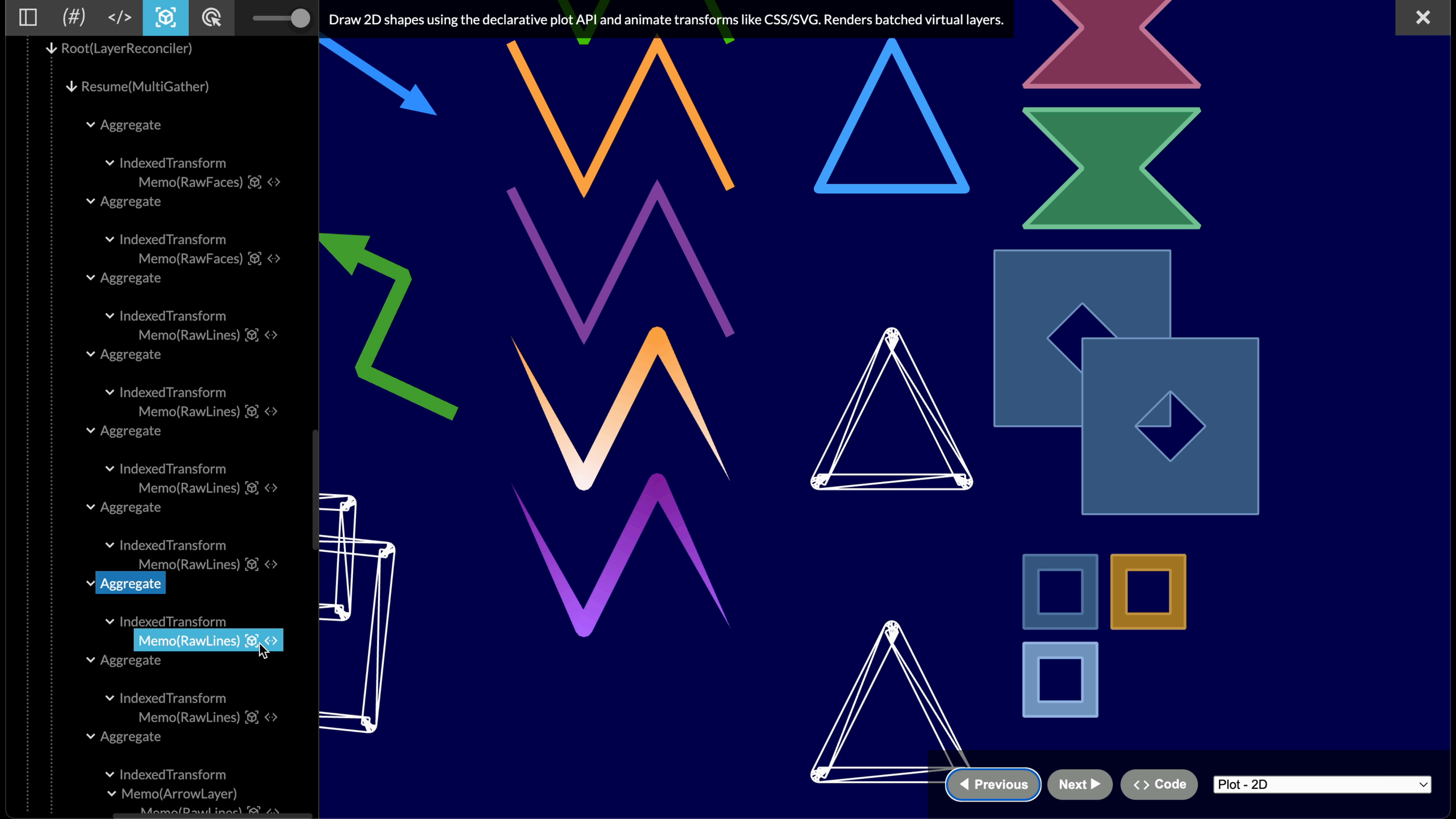

Draw 2D shapes using the declarative plot API and animate transforms like CSS/SVG. Renders batched virtual layers.
Root(LayerReconciler)
Resume(MultiGather)
Aggregate
IndexedTransform
Memo(RawFaces)
Aggregate
IndexedTransform
Memo(RawFaces)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(ArrowLayer)
Previous
Next
Code
Plot - 2D

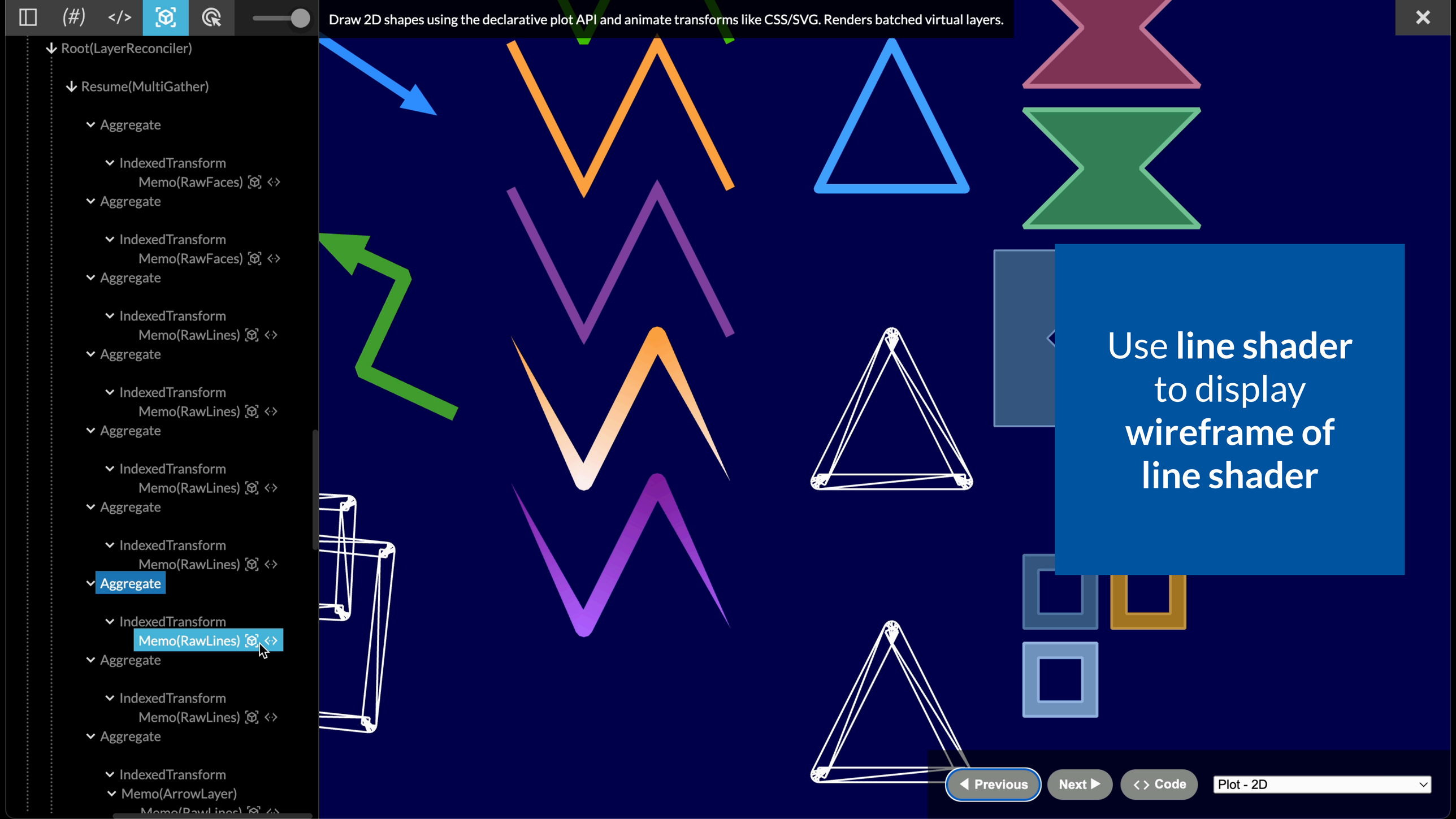

Draw 2D shapes using the declarative plot API and animate transforms like CSS/SVG. Renders batched virtual layers.
Root(LayerReconciler)
Resume(MultiGather)
Aggregate
IndexedTransform
Memo(RawFaces)
Aggregate
IndexedTransform
Memo(RawFaces)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(ArrowLayer)
Previous
Next
Code
Plot - 2D

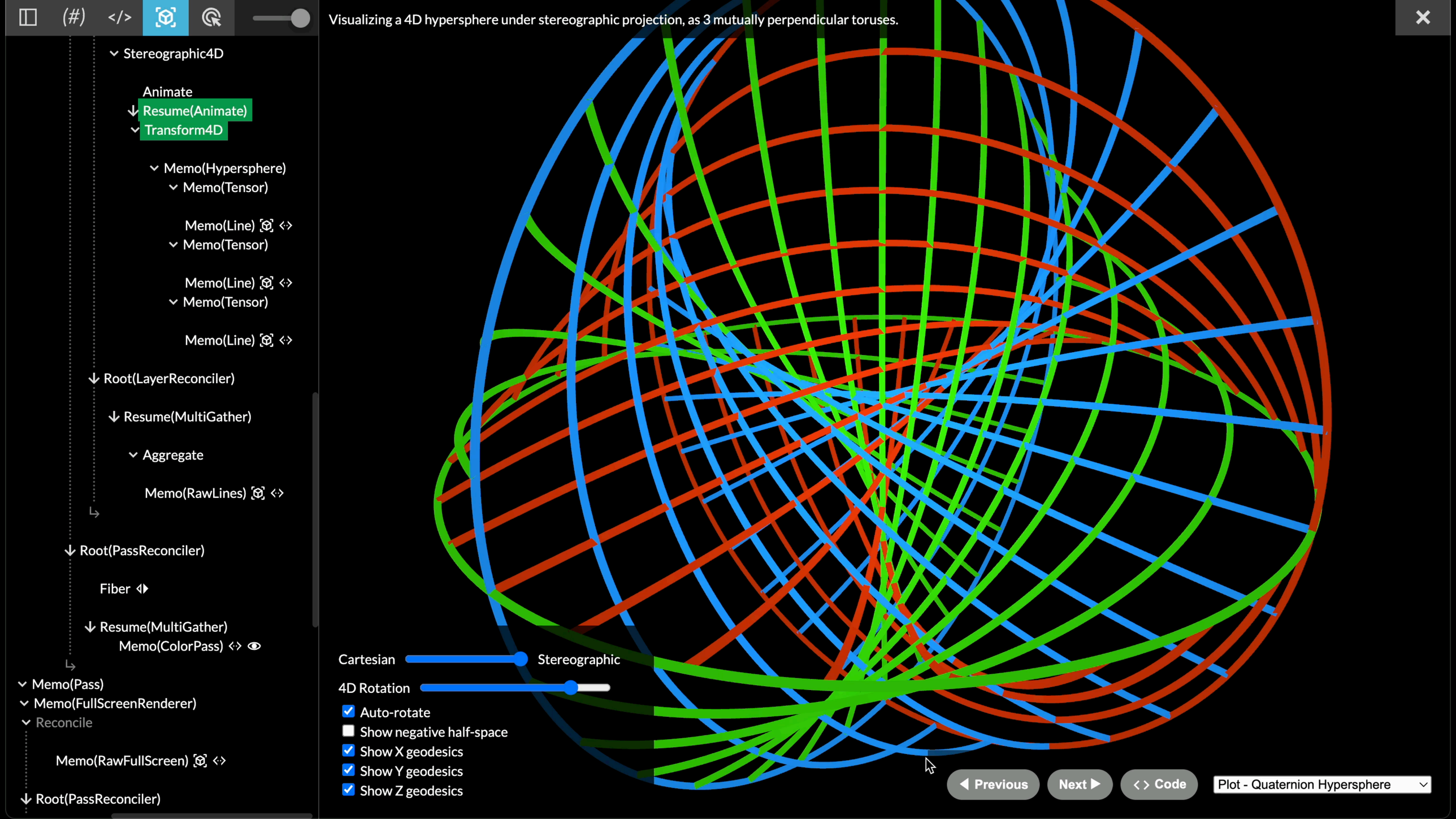

Draw 2D shapes using the declarative plot API and animate transforms like CSS/SVG. Renders batched virtual layers.
Root(LayerReconciler)
Resume(MultiGather)
Aggregate
IndexedTransform
Memo(RawFaces)
Aggregate
IndexedTransform
Memo(RawFaces)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(ArrowLayer)
Every layer
=
unique
shader
permutation
Previous
Next
Code
Plot - 2D

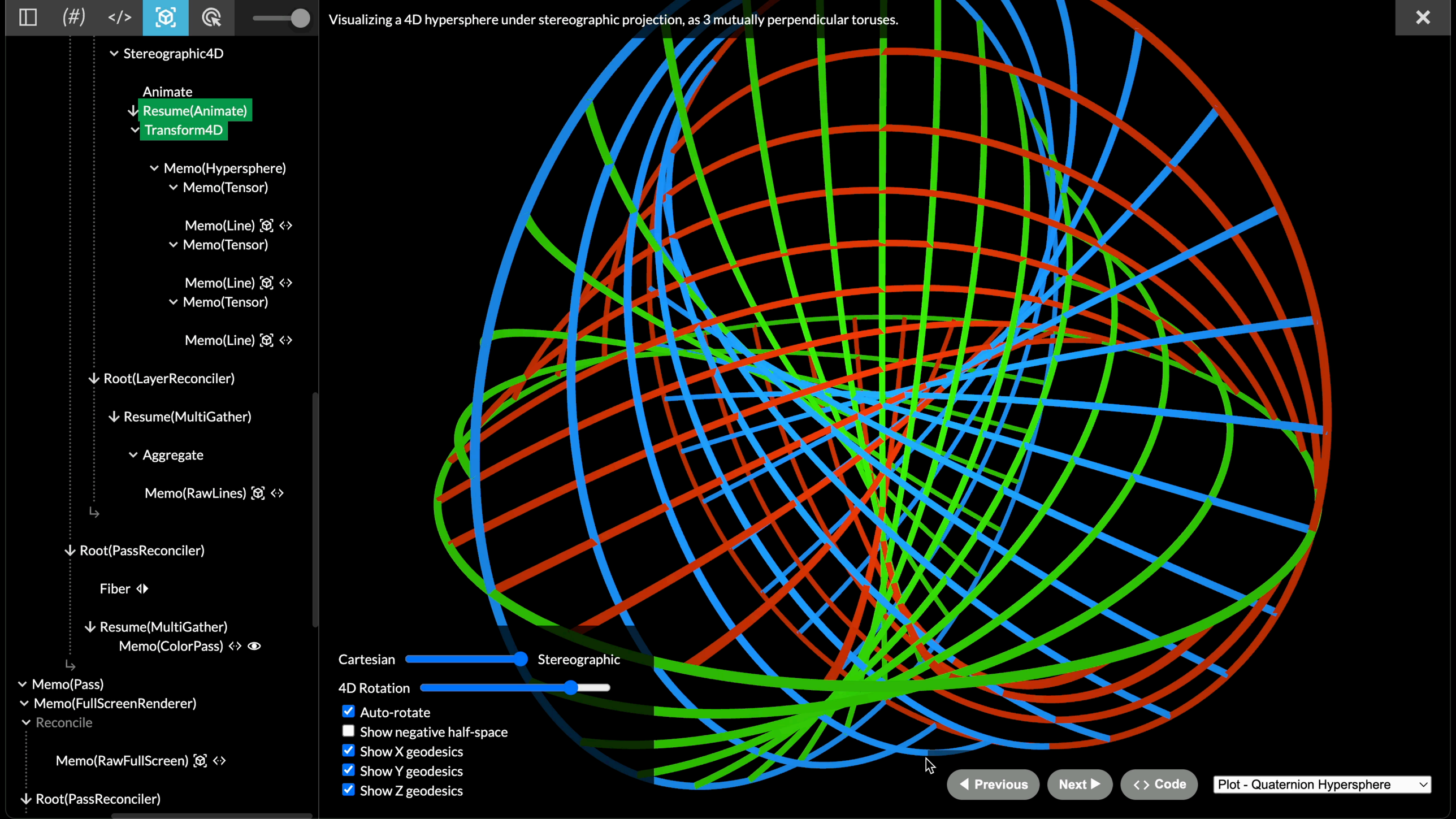

Draw 2D shapes using the declarative plot API and animate transforms like CSS/SVG. Renders batched virtual layers.
Root(LayerReconciler)
Resume(MultiGather)
Aggregate
IndexedTransform
Memo(RawFaces)
Aggregate
IndexedTransform
Memo(RawFaces)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(ArrowLayer)
Every layer = unique shader permutation
Highlight = new pipeline!
Previous
Next
Code
Plot - 2D

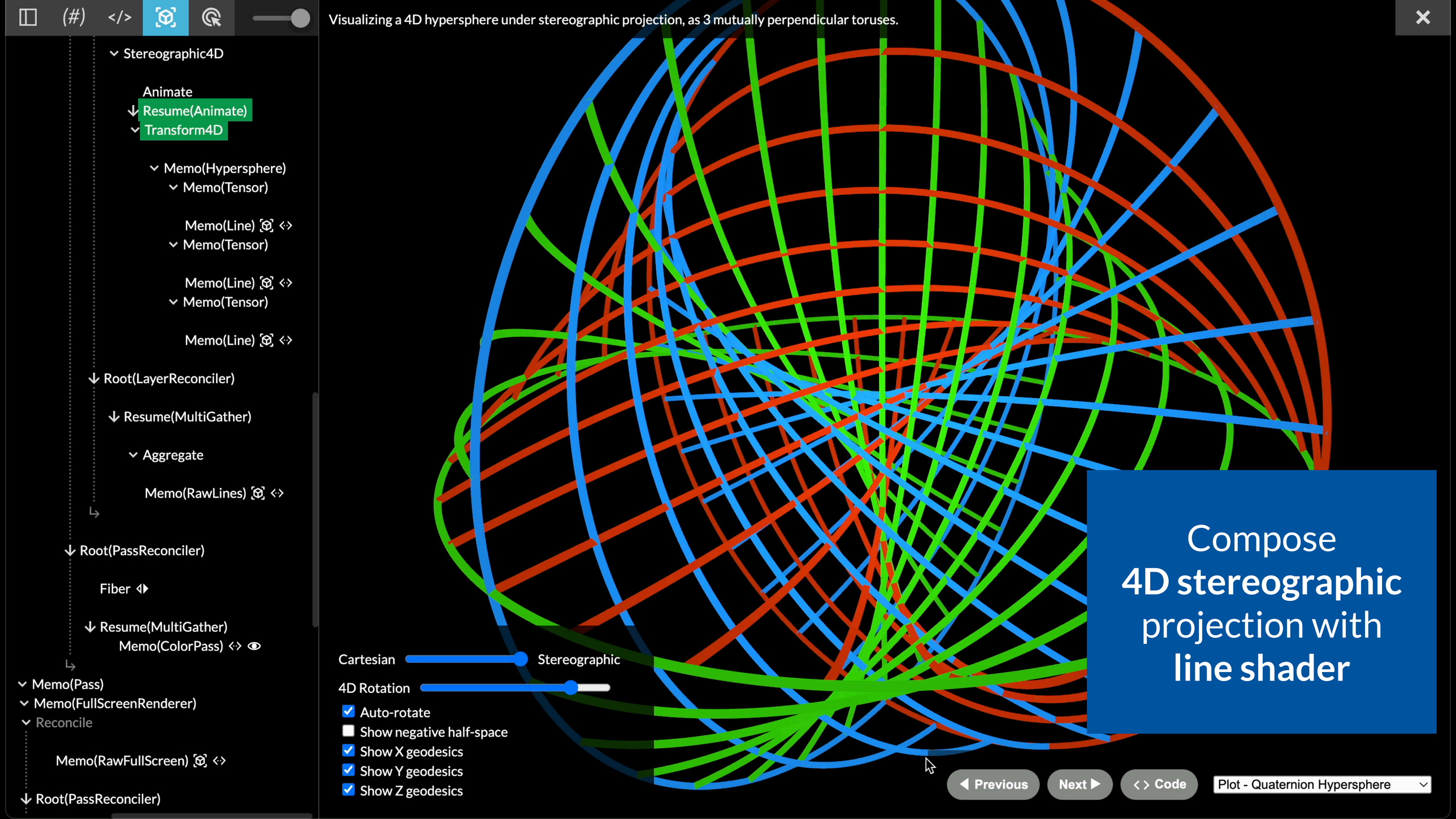

Draw 2D shapes using the declarative plot API and animate transforms like CSS/SVG. Renders batched virtual layers.
Root(LayerReconciler)
Resume(MultiGather)
Aggregate
IndexedTransform
Memo(RawFaces)
Aggregate
IndexedTransform
Memo(RawFaces)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(ArrowLayer)
Previous
Next
Code
Plot - 2D

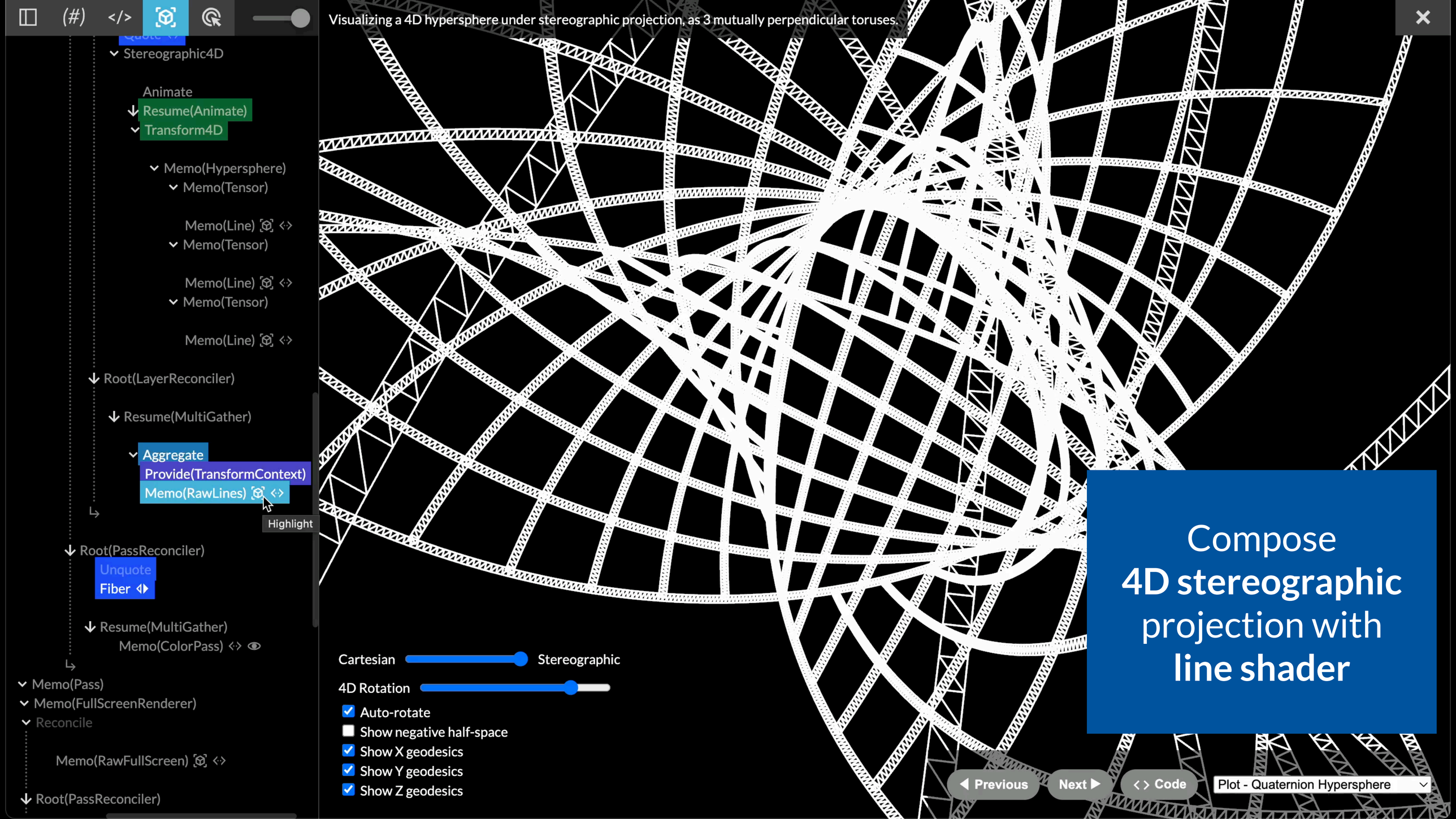

Draw 2D shapes using the declarative plot API and animate transforms like CSS/SVG. Renders batched virtual layers.
Root(LayerReconciler)
Resume(MultiGather)
Aggregate
IndexedTransform
Memo(RawFaces)
Aggregate
IndexedTransform
Memo(RawFaces)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(ArrowLayer)
Previous
Next
Code
Plot - 2D

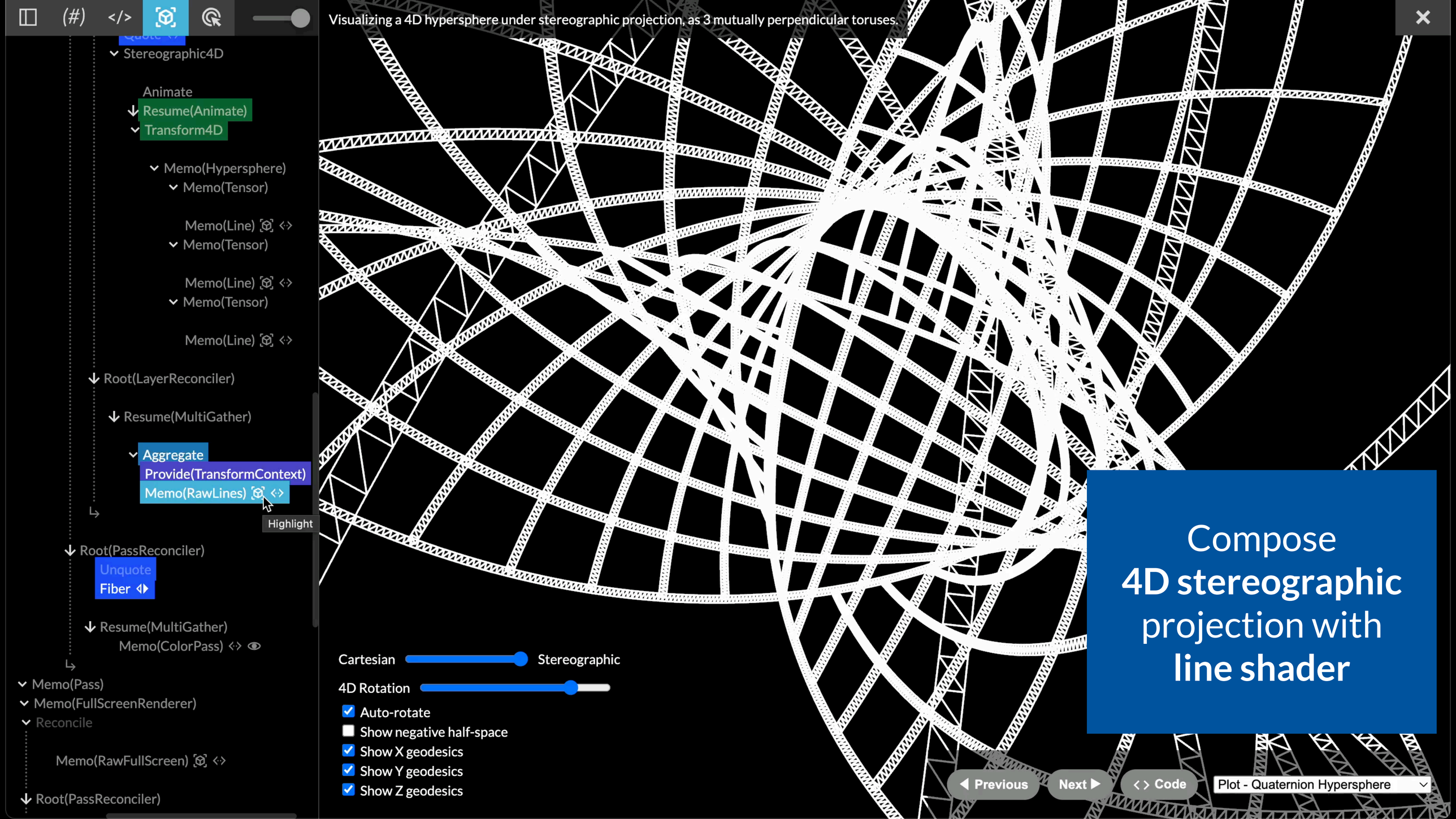

Draw 2D shapes using the declarative plot API and animate transforms like CSS/SVG. Renders batched virtual layers.
Root(LayerReconciler)
Resume(MultiGather)
Aggregate
IndexedTransform
Memo(RawFaces)
Aggregate
IndexedTransform
Memo(RawFaces)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(RawLines)
Aggregate
IndexedTransform
Memo(ArrowLayer)
Use line shader to display wireframe of line shader
Previous
Next
Code
Plot - 2D

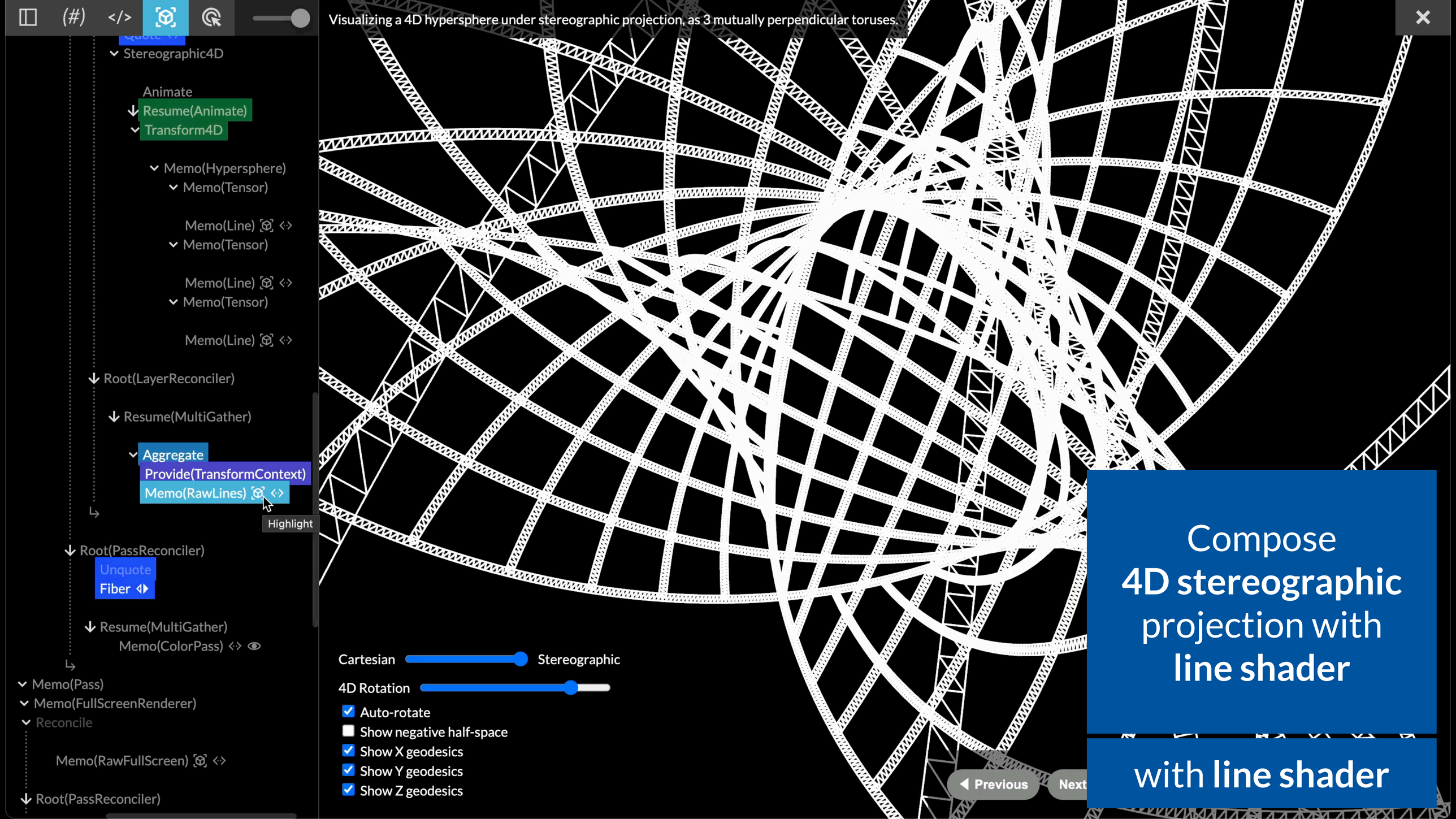

Visualizing a 4D hypersphere under stereographic projection, as 3 mutually perpendicular toruses.
Stereographic4D
Animate
Resume(Animate)
Transform4D
Memo(Hypersphere)
Memo(Tensor)
Memo(Line)
Memo(Tensor)
Memo(Line)
Memo(Tensor)
Memo(Line)
Root(LayerReconciler)
Resume(MultiGather)
Aggregate
Memo(RawLines)
Root(PassReconciler)
Fiber
Resume(MultiGather)
Memo(ColorPass)
Memo(Pass)
Memo(FullScreenRenderer)
Reconcile
Memo(RawFullScreen)
Root(PassReconciler)
Cartesian
Stereographic
4D Rotation
Auto-rotate
Show negative half-space
Show X geodesics
Show Y geodesics
Show Z geodesics
Previous
Next
Code
Plot - Quaternion Hypersphere

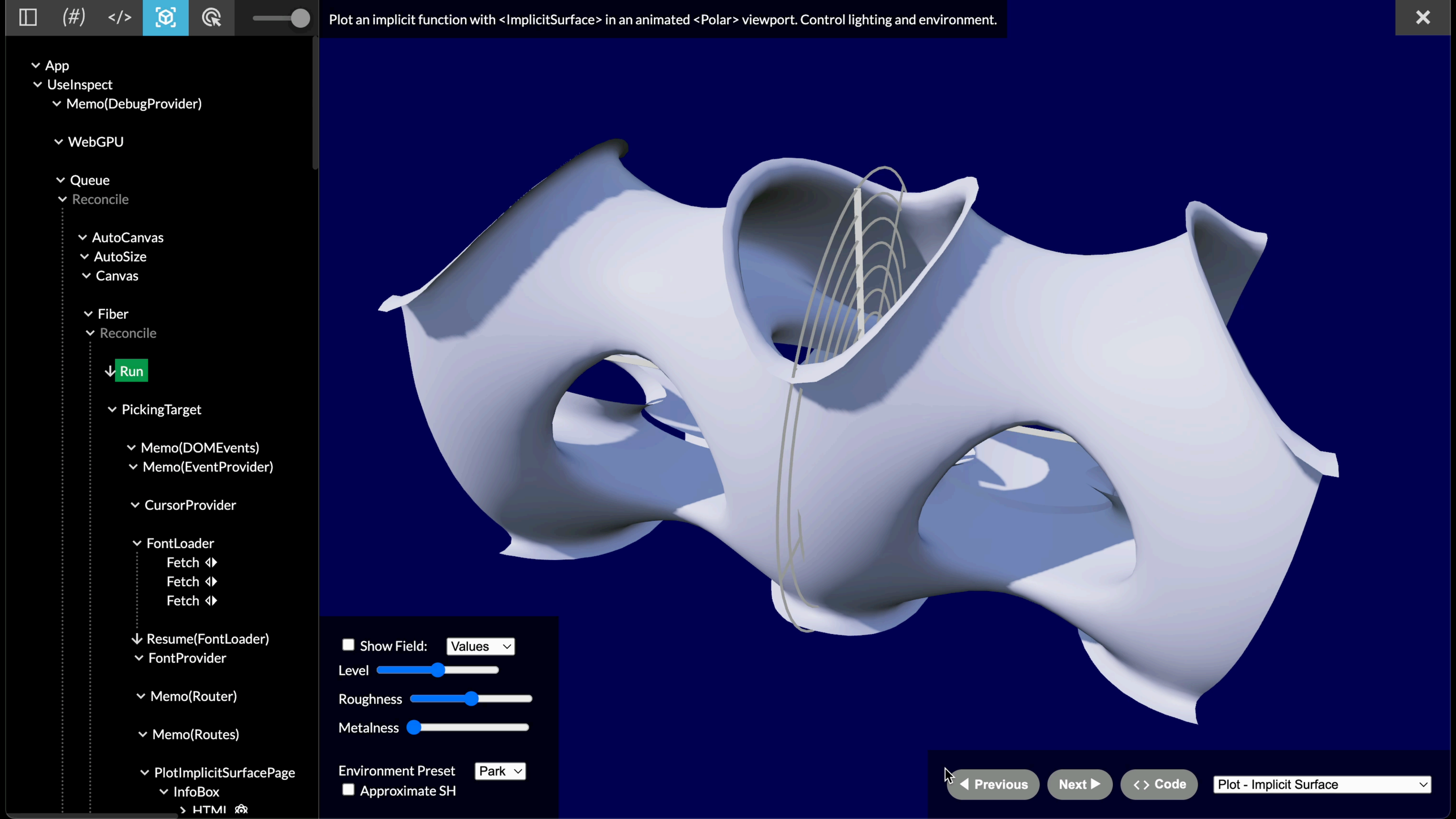

Visualizing a 4D hypersphere under stereographic projection, as 3 mutually perpendicular toruses.
Stereographic4D
Animate
Resume(Animate)
Transform4D
Memo(Hypersphere)
Memo(Tensor)
Memo(Line)
Memo(Tensor)
Memo(Line)
Memo(Tensor)
Memo(Line)
Root(LayerReconciler)
Resume(MultiGather)
Aggregate
Memo(RawLines)
Root(PassReconciler)
Fiber
Resume(MultiGather)
Memo(ColorPass)
Memo(Pass)
Memo(FullScreenRenderer)
Reconcile
Memo(RawFullScreen)
Root(PassReconciler)
Cartesian
Stereographic
4D Rotation
Auto-rotate
Show negative half-space
Show X geodesics
Show Y geodesics
Show Z geodesics
Previous
Next
Code
Plot - Quaternion Hypersphere

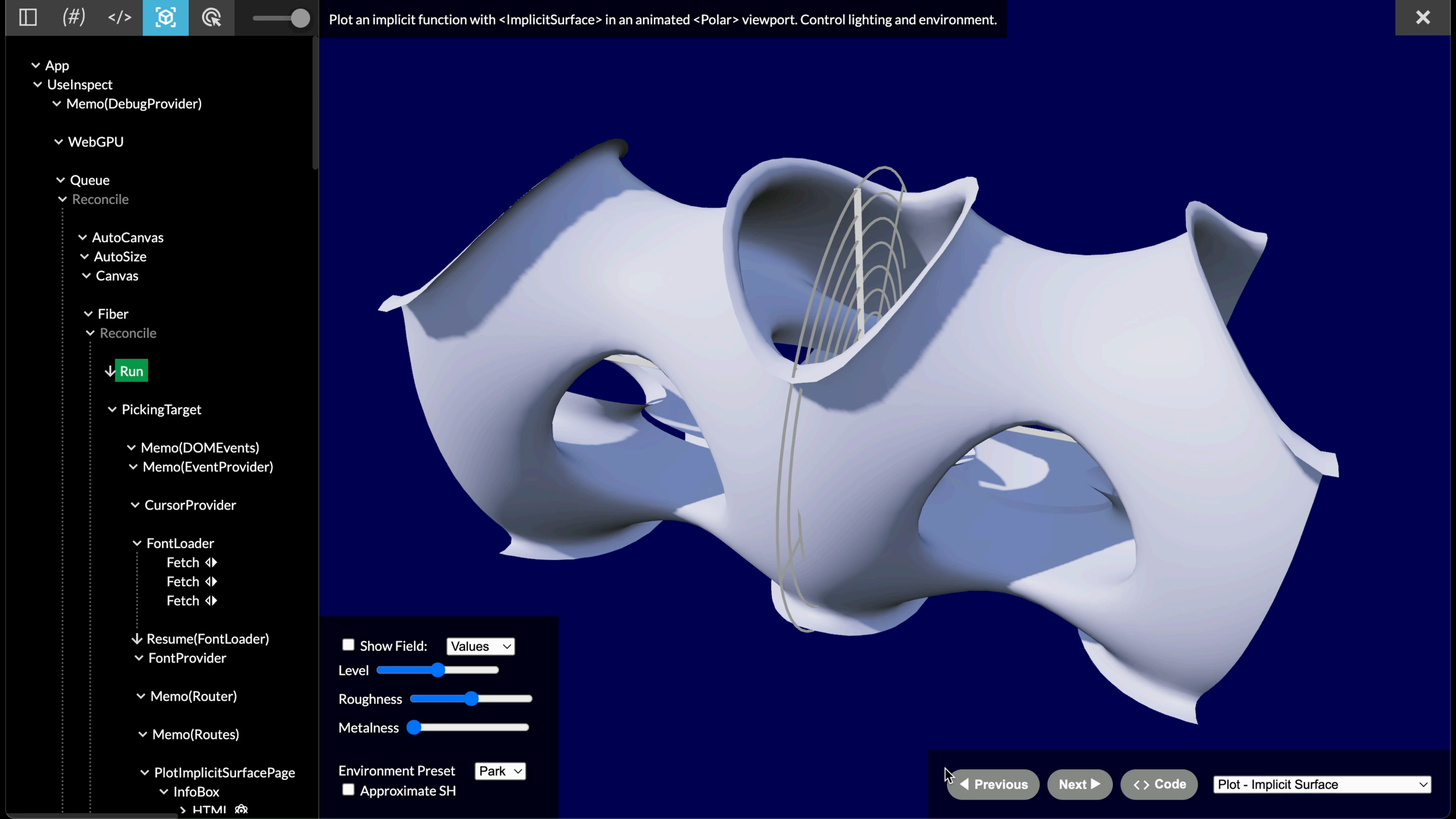

Visualizing a 4D hypersphere under stereographic projection, as 3 mutually perpendicular toruses.
Stereographic4D
Animate
Resume(Animate)
Transform4D
Memo(Hypersphere)
Memo(Tensor)
Memo(Line)
Memo(Tensor)
Memo(Line)
Memo(Tensor)
Memo(Line)
Root(LayerReconciler)
Resume(MultiGather)
Aggregate
Memo(RawLines)
Root(PassReconciler)
Fiber
Resume(MultiGather)
Memo(ColorPass)
Memo(Pass)
Memo(FullScreenRenderer)
Reconcile
Memo(RawFullScreen)
Root(PassReconciler)
Cartesian
Stereographic
4D Rotation
Auto-rotate
Show negative half-space
Show X geodesics
Show Y geodesics
Show Z geodesics
Compose
4D stereographic
projection with
line shader
Previous
Next
Code
Plot - Quaternion Hypersphere

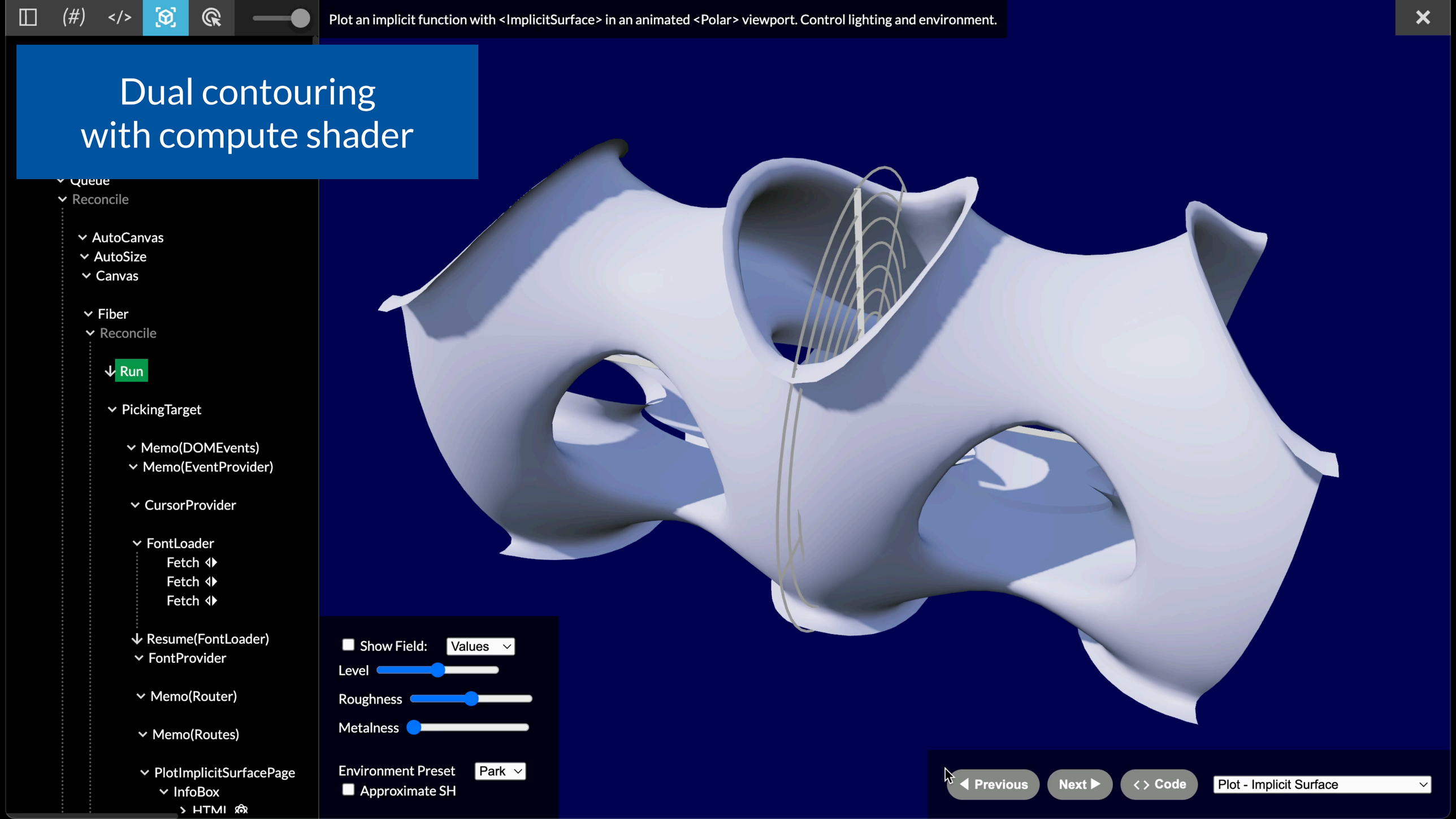

Visualizing a 4D hypersphere under stereographic projection, as 3 mutually perpendicular toruses.
Stereographic4D
Animate
Resume(Animate)
Transform4D
Memo(Hypersphere)
Memo(Tensor)
Memo(Line)
Memo(Tensor)
Memo(Line)
Memo(Tensor)
Memo(Line)
Root(LayerReconciler)
Resume(MultiGather)
Aggregate
Provide(TransformContext)
Memo(RawLines)
Highlight
Root(PassReconciler)
Unquote
Fiber
Resume(MultiGather)
Memo(ColorPass)
Memo(Pass)
Memo(FullScreenRenderer)
Reconcile
Memo(RawFullScreen)
Root(PassReconciler)
Cartesian
Stereographic
4D Rotation
Auto-rotate
Show negative half-space
Show X geodesics
Show Y geodesics
Show Z geodesics
Compose
4D stereographic
projection with
line shader
Previous
Next
Code
Plot - Quaternion Hypersphere

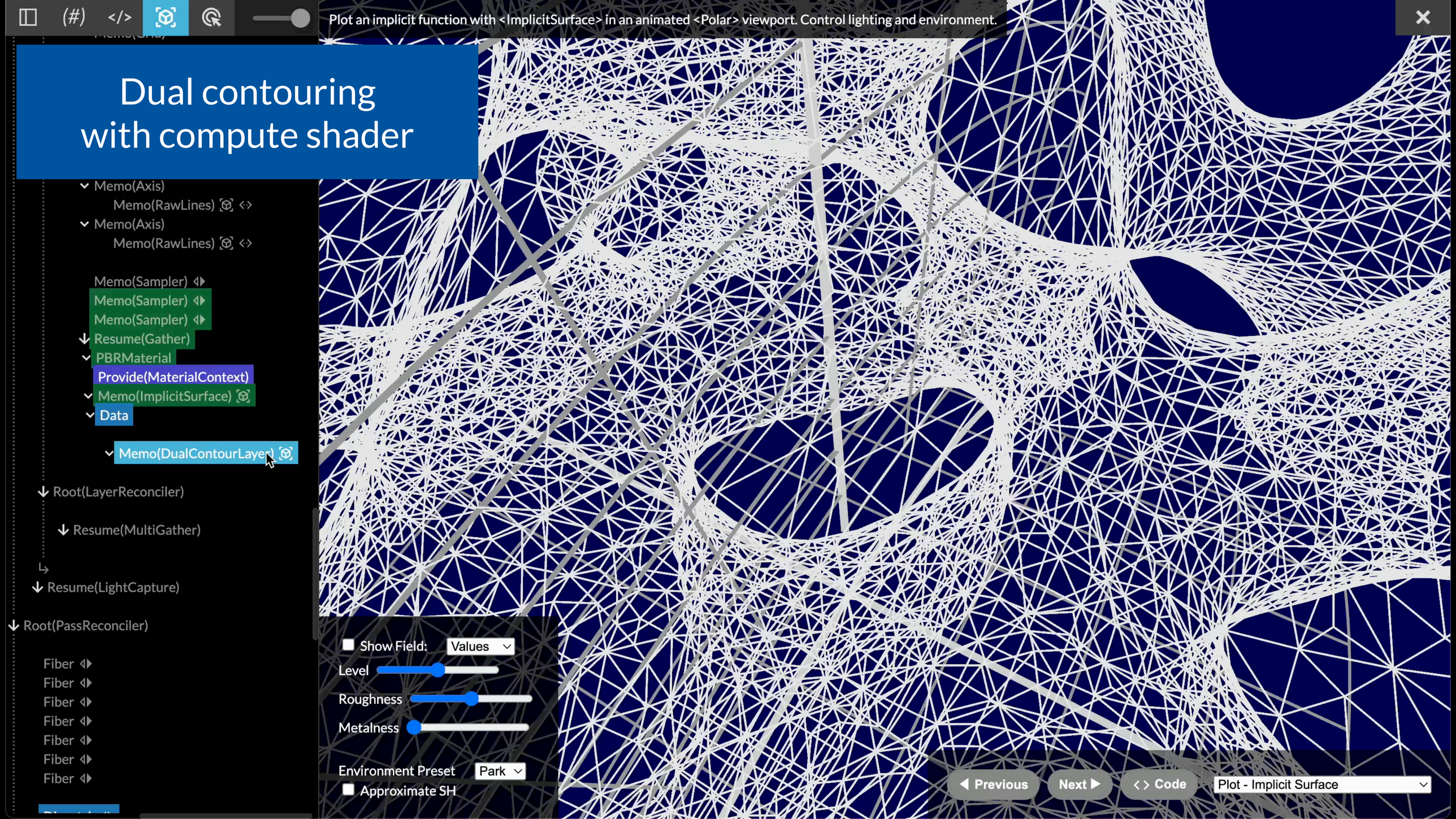

Visualizing a 4D hypersphere under stereographic projection, as 3 mutually perpendicular toruses.
Stereographic4D
Animate
Resume(Animate)
Transform4D
Memo(Hypersphere)
Memo(Tensor)
Memo(Line)
Memo(Tensor)
Memo(Line)
Memo(Tensor)
Memo(Line)
Root(LayerReconciler)
Resume(MultiGather)
Aggregate
Provide(TransformContext)
Memo(RawLines)
Highlight
Root(PassReconciler)
Unquote
Fiber
Resume(MultiGather)
Memo(ColorPass)
Memo(Pass)
Memo(FullScreenRenderer)
Reconcile
Memo(RawFullScreen)
Root(PassReconciler)
Cartesian
Stereographic
4D Rotation
Auto-rotate
Show negative half-space
Show X geodesics
Show Y geodesics
Show Z geodesics
Compose
4D stereographic
projection with
line shader
Previous
Next
Code
Plot - Quaternion Hypersphere

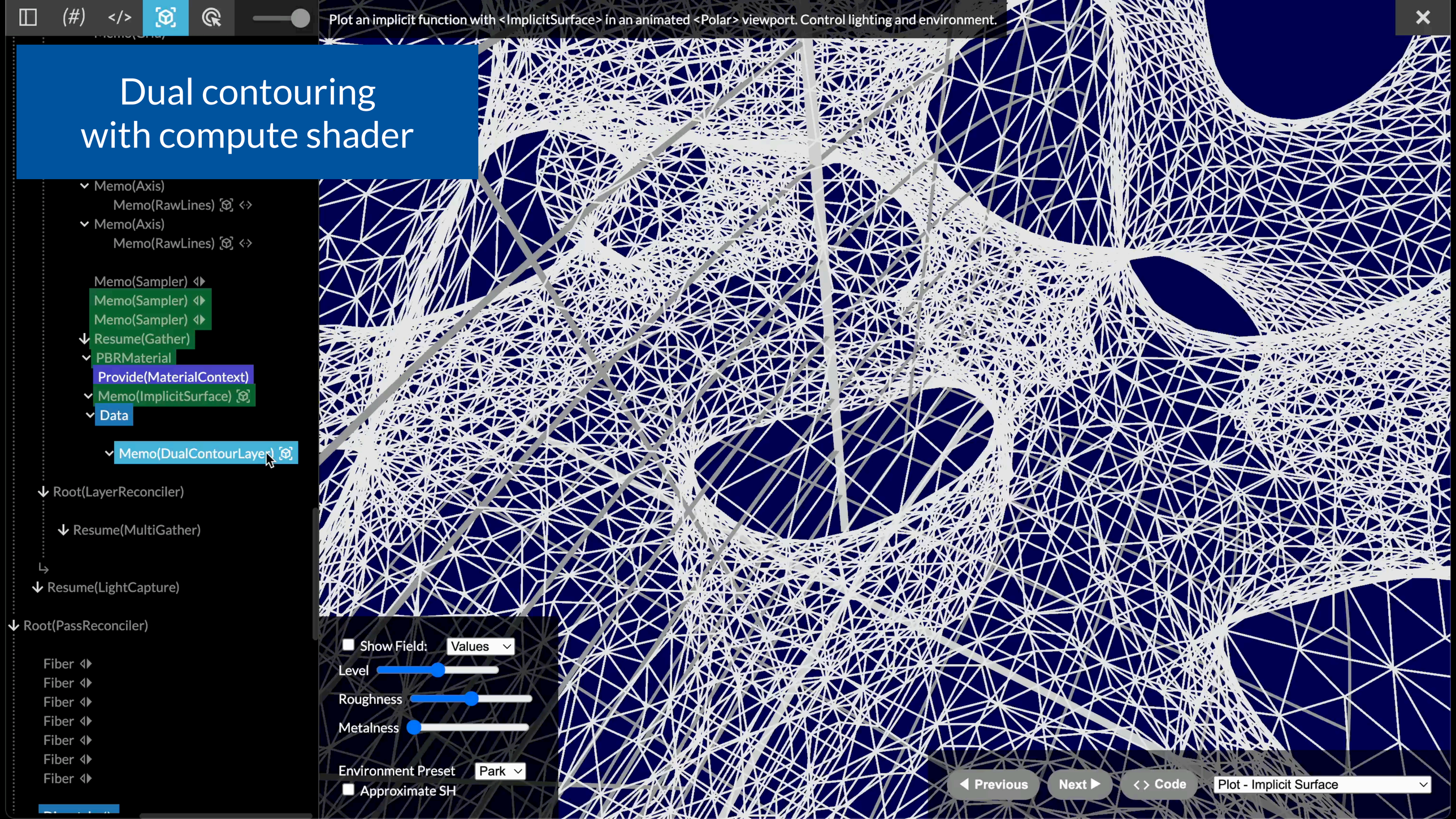

Visualizing a 4D hypersphere under stereographic projection, as 3 mutually perpendicular toruses.
Stereographic4D
Animate
Resume(Animate)
Transform4D
Memo(Hypersphere)
Memo(Tensor)
Memo(Line)
Memo(Tensor)
Memo(Line)
Memo(Tensor)
Memo(Line)
Root(LayerReconciler)
Resume(MultiGather)
Aggregate
Provide(TransformContext)
Memo(RawLines)
Highlight
Root(PassReconciler)
Unquote
Fiber
Resume(MultiGather)
Memo(ColorPass)
Memo(Pass)
Memo(FullScreenRenderer)
Reconcile
Memo(RawFullScreen)
Root(PassReconciler)
Cartesian
Stereographic
4D Rotation
Auto-rotate
Show negative half-space
Show X geodesics
Show Y geodesics
Show Z geodesics
Previous
Next
Compose
4D stereographic
projection with
line shader
with line shader

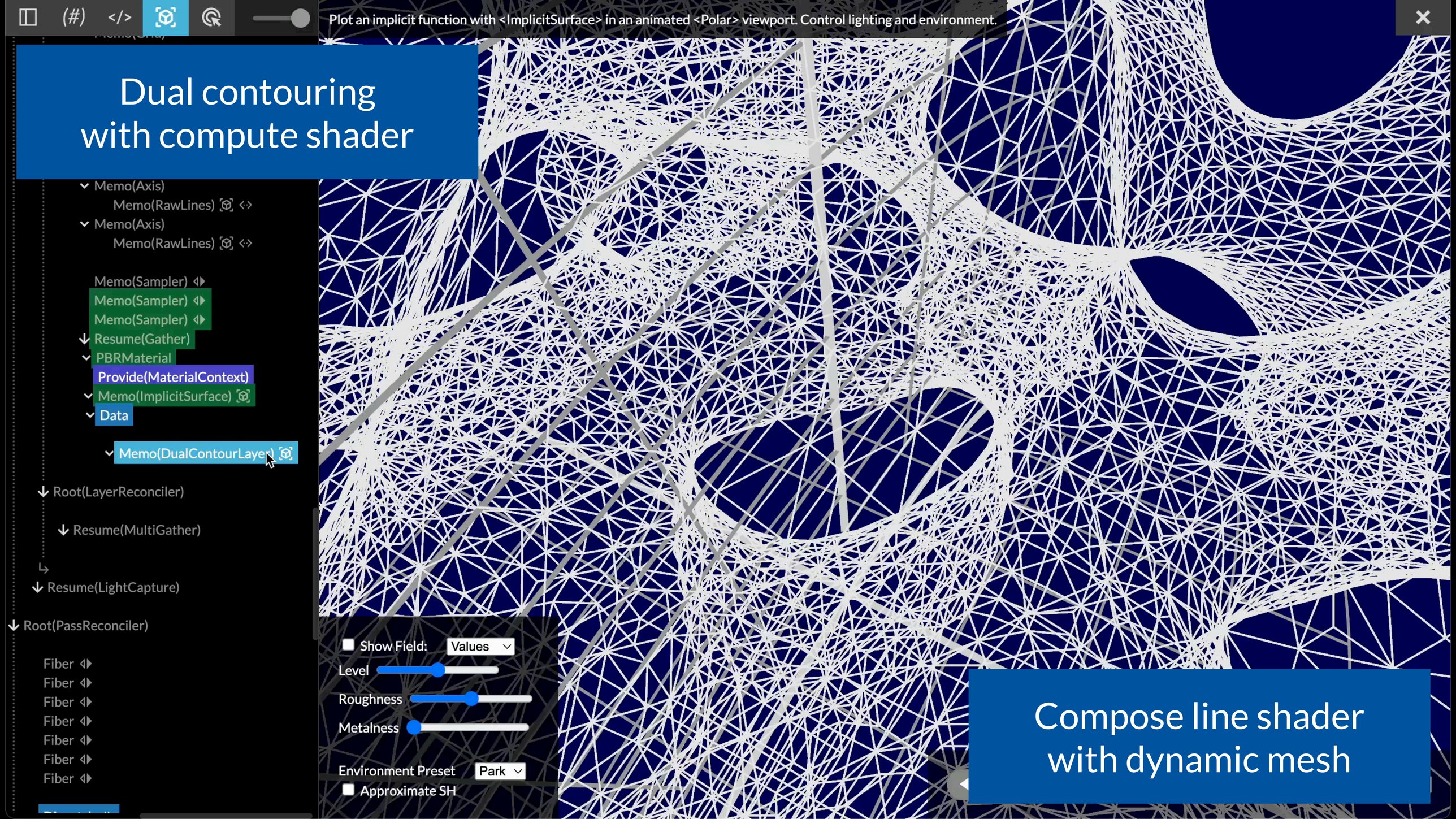

Plot an implicit function with <ImplicitSurface> in an animated <Polar> viewport. Control lighting and environment.
App
UseInspect
Memo(DebugProvider)
WebGPU
Queue
Reconcile
AutoCanvas
AutoSize
Canvas
Fiber
Reconcile
Run
PickingTarget
Memo(DOMEvents)
Memo(EventProvider)
CursorProvider
FontLoader
Fetch
Fetch
Fetch
Resume(FontLoader)
FontProvider
Memo(Router)
Memo(Routes)
PlotImplicitSurfacePage
InfoBox
HTML
Show Field:
Values
Level
Roughness
Metalness
Environment Preset
Park
Approximate SH
Previous
Next
Code
Plot - Implicit Surface

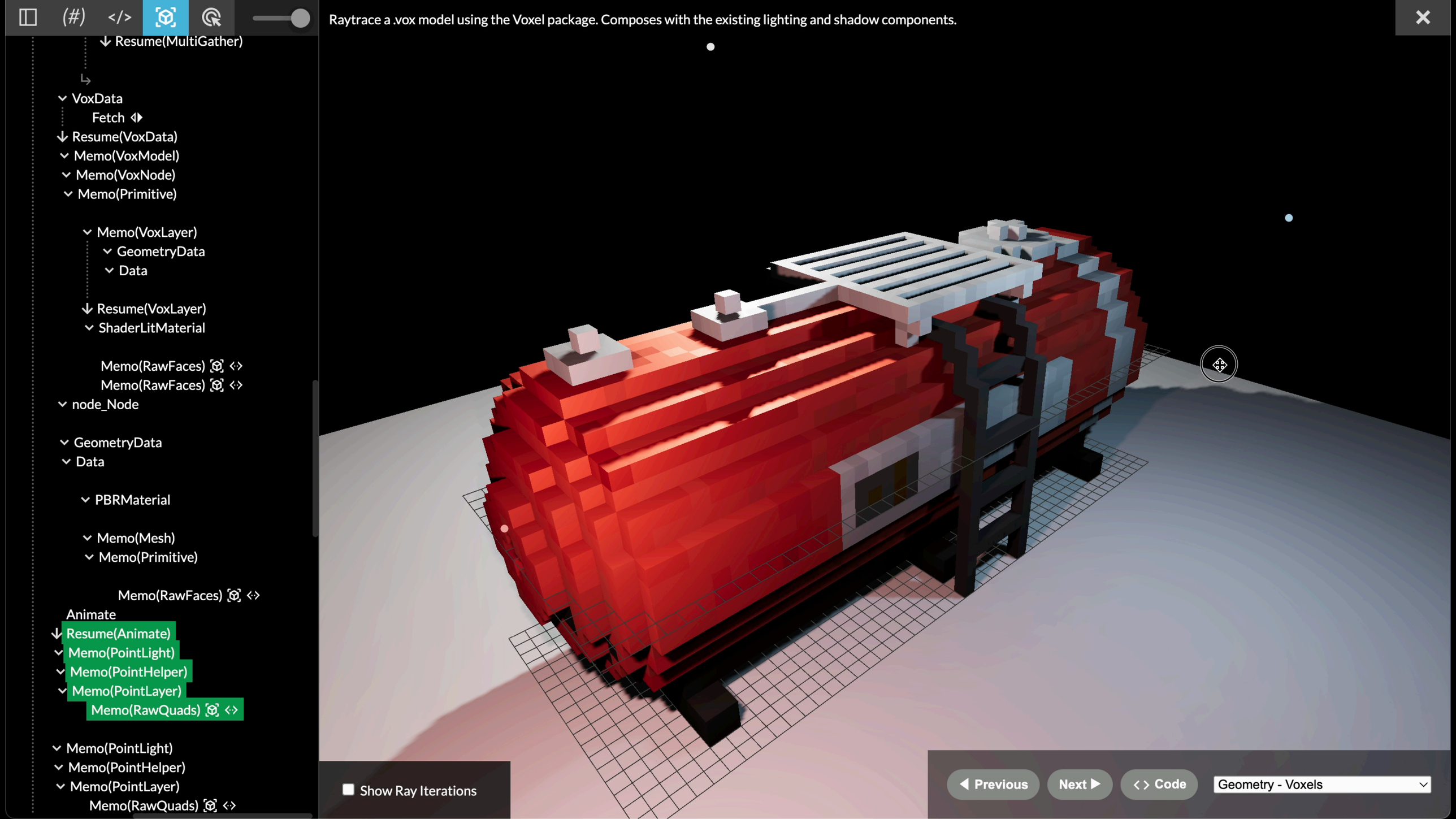

Plot an implicit function with <ImplicitSurface> in an animated <Polar> viewport. Control lighting and environment.
App
UseInspect
Memo(DebugProvider)
WebGPU
Queue
Reconcile
AutoCanvas
AutoSize
Canvas
Fiber
Reconcile
Run
PickingTarget
Memo(DOMEvents)
Memo(EventProvider)
CursorProvider
FontLoader
Fetch
Fetch
Fetch
Resume(FontLoader)
FontProvider
Memo(Router)
Memo(Routes)
PlotImplicitSurfacePage
InfoBox
HTML
Show Field:
Values
Level
Roughness
Metalness
Environment Preset
Park
Approximate SH
Previous
Next
Code
Plot - Implicit Surface

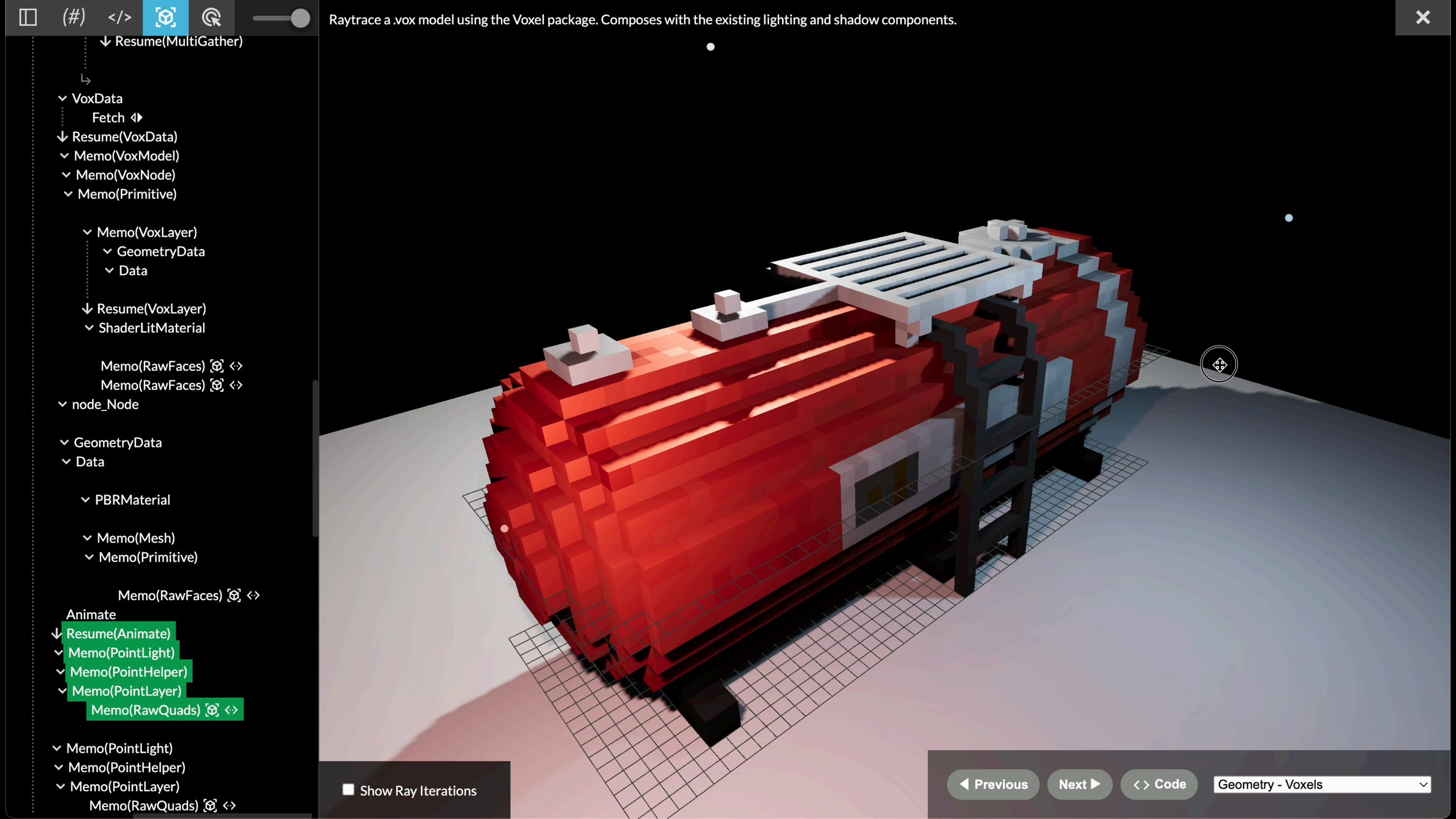

Plot an implicit function with <ImplicitSurface> in an animated <Polar> viewport. Control lighting and environment.
Dual contouring
with compute shader
Queue
Reconcile
AutoCanvas
AutoSize
Canvas
Fiber
Reconcile
Run
PickingTarget
Memo(DOMEvents)
Memo(EventProvider)
CursorProvider
FontLoader
Fetch
Fetch
Fetch
Resume(FontLoader)
FontProvider
Memo(Router)
Memo(Routes)
PlotImplicitSurfacePage
InfoBox
HTML
Show Field:
Values
Level
Roughness
Metalness
Environment Preset
Park
Approximate SH
Previous
Next
Code
Plot - Implicit Surface

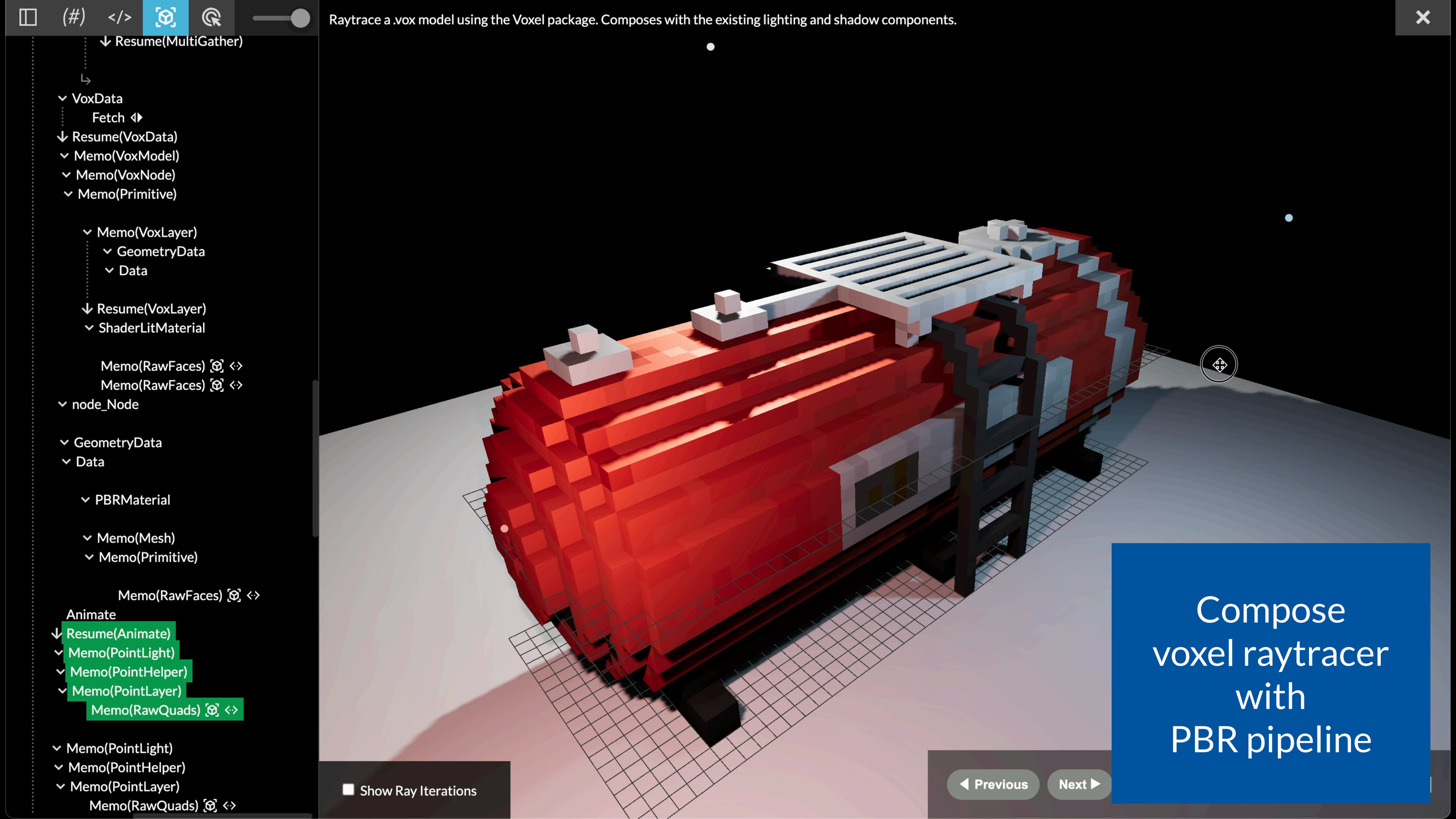

Plot an implicit function with <ImplicitSurface> in an animated <Polar> viewport. Control lighting and environment.
Dual contouring
with compute shader
Memo(Axis)
Memo(RawLines)
Memo(Axis)
Memo(RawLines)
Memo(Sampler)
Memo(Sampler)
Memo(Sampler)
Resume(Gather)
PBRMaterial
Provide(MaterialContext)
Memo(ImplicitSurface)
Data
Memo(DualContourLayer)
Root(LayerReconciler)
Resume(MultiGather)
Resume(LightCapture)
Root(PassReconciler)
Fiber
Fiber
Fiber
Fiber
Fiber
Fiber
Fiber
Show Field:
Values
Level
Roughness
Metalness
Environment Preset
Park
Approximate SH
Previous
Next
Code
Plot - Implicit Surface

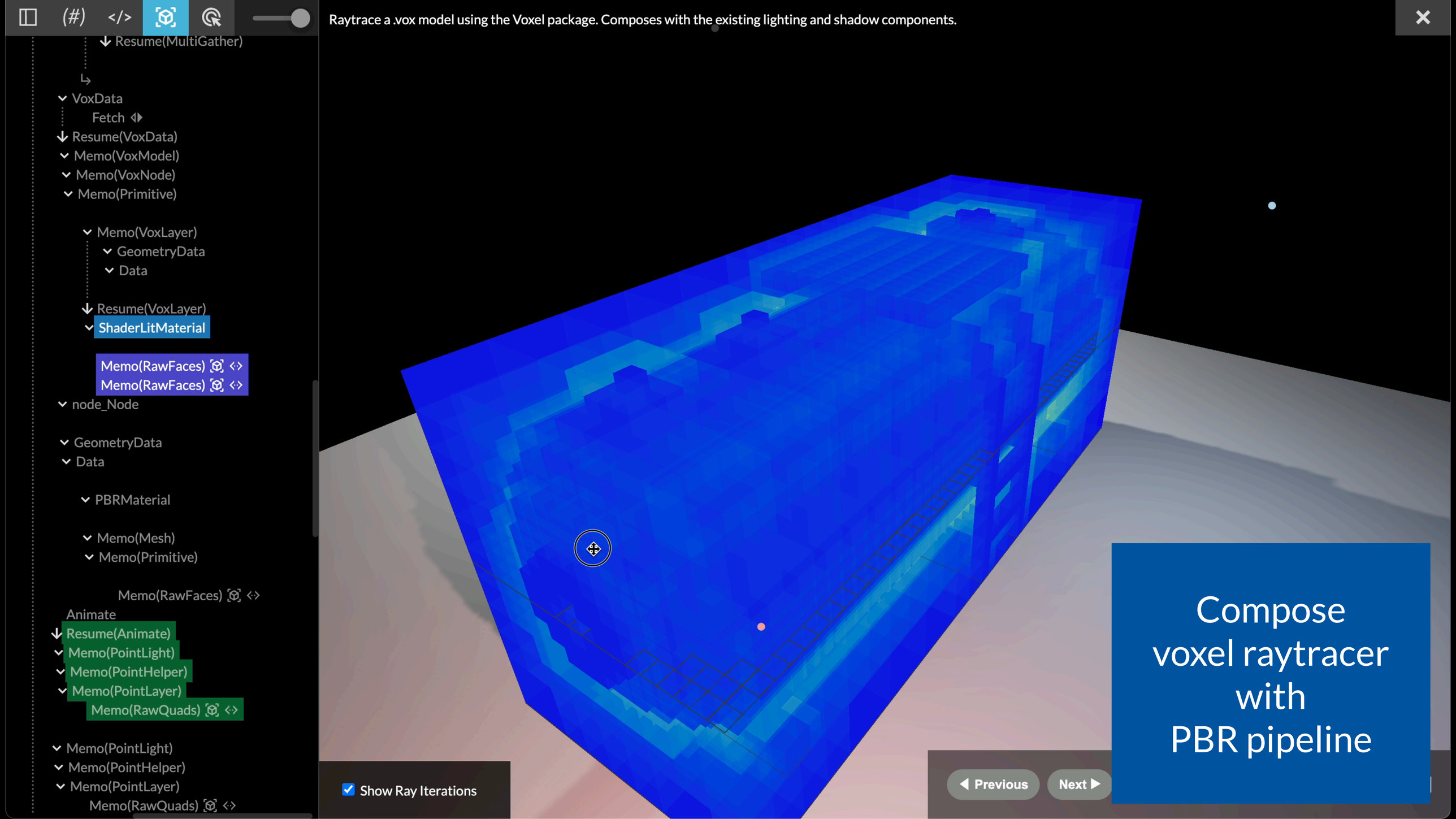

Plot an implicit function with <ImplicitSurface> in an animated <Polar> viewport. Control lighting and environment.
Dual contouring
with compute shader
Memo(Axis)
Memo(RawLines)
Memo(Axis)
Memo(RawLines)
Memo(Sampler)
Memo(Sampler)
Memo(Sampler)
Resume(Gather)
PBRMaterial
Provide(MaterialContext)
Memo(ImplicitSurface)
Data
Memo(DualContourLayer)
Root(LayerReconciler)
Resume(MultiGather)
Resume(LightCapture)
Root(PassReconciler)
Fiber
Fiber
Fiber
Fiber
Fiber
Fiber
Fiber
Show Field:
Values
Level
Roughness
Metalness
Environment Preset
Park
Approximate SH
Previous
Next
Code
Plot - Implicit Surface

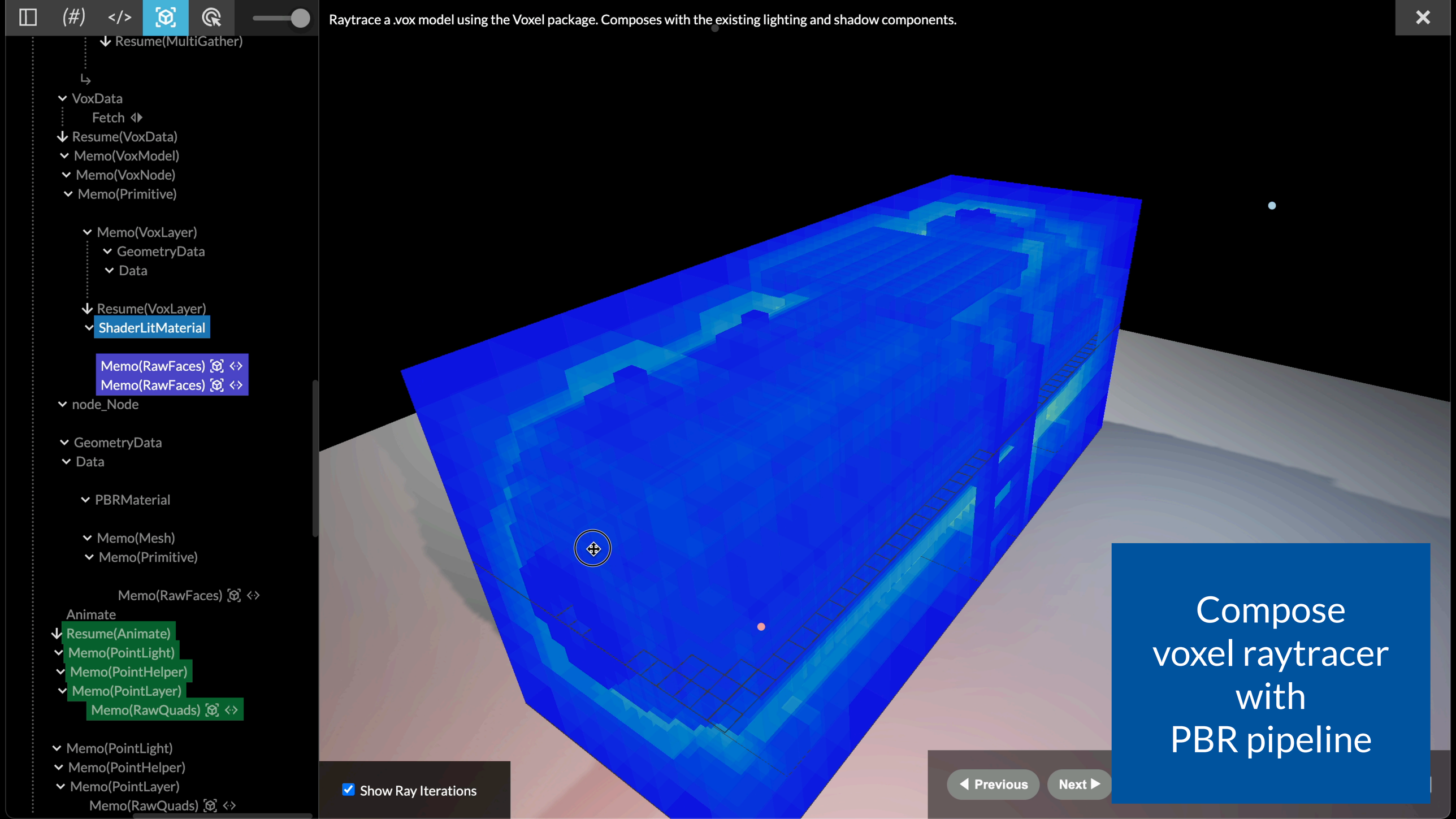

Plot an implicit function with <ImplicitSurface> in an animated <Polar> viewport. Control lighting and environment.
Dual contouring
with compute shader
Memo(Axis)
Memo(RawLines)
Memo(Axis)
Memo(RawLines)
Memo(Sampler)
Memo(Sampler)
Memo(Sampler)
Resume(Gather)
PBRMaterial
Provide(MaterialContext)
Memo(ImplicitSurface)
Data
Memo(DualContourLayer)
Root(LayerReconciler)
Resume(MultiGather)
Resume(LightCapture)
Root(PassReconciler)
Fiber
Fiber
Fiber
Fiber
Fiber
Fiber
Fiber
Show Field:
Values
Level
Roughness
Metalness
Environment Preset
Park
Approximate SH
Compose line shader
with dynamic mesh


Raytrace a .vox model using the Voxel package. Composes with the existing lighting and shadow components.
Resume(MultiGather)
VoxData
Fetch
Resume(VoxData)
Memo(VoxModel)
Memo(VoxNode)
Memo(Primitive)
Memo(VoxLayer)
GeometryData
Data
Resume(VoxLayer)
ShaderLitMaterial
Memo(RawFaces)
Memo(RawFaces)
node_Node
GeometryData
Data
PBRMaterial
Memo(Mesh)
Memo(Primitive)
Memo(RawFaces)
Animate
Resume(Animate)
Memo(PointLight)
Memo(PointHelper)
Memo(PointLayer)
Memo(RawQuads)
Memo(PointLight)
Memo(PointHelper)
Memo(PointLayer)
Memo(RawQuads)
Show Ray Iterations
Previous
Next
Code
Geometry - Voxels


Raytrace a .vox model using the Voxel package. Composes with the existing lighting and shadow components.
Resume(MultiGather)
VoxData
Fetch
Resume(VoxData)
Memo(VoxModel)
Memo(VoxNode)
Memo(Primitive)
Memo(VoxLayer)
GeometryData
Data
Resume(VoxLayer)
ShaderLitMaterial
Memo(RawFaces)
Memo(RawFaces)
node_Node
GeometryData
Data
PBRMaterial
Memo(Mesh)
Memo(Primitive)
Memo(RawFaces)
Animate
Resume(Animate)
Memo(PointLight)
Memo(PointHelper)
Memo(PointLayer)
Memo(RawQuads)
Memo(PointLight)
Memo(PointHelper)
Memo(PointLayer)
Memo(RawQuads)
Show Ray Iterations
Previous
Next
Code
Geometry - Voxels


Raytrace a .vox model using the Voxel package. Composes with the existing lighting and shadow components.
Resume(MultiGather)
VoxData
Fetch
Resume(VoxData)
Memo(VoxModel)
Memo(VoxNode)
Memo(Primitive)
Memo(VoxLayer)
GeometryData
Data
Resume(VoxLayer)
ShaderLitMaterial
Memo(RawFaces)
Memo(RawFaces)
node_Node
GeometryData
Data
PBRMaterial
Memo(Mesh)
Memo(Primitive)
Memo(RawFaces)
Animate
Resume(Animate)
Memo(PointLight)
Memo(PointHelper)
Memo(PointLayer)
Memo(RawQuads)
Memo(PointLight)
Memo(PointHelper)
Memo(PointLayer)
Memo(RawQuads)
Show Ray Iterations
Previous
Next
Compose voxel raytracer with PBR pipeline


Raytrace a .vox model using the Voxel package. Composes with the existing lighting and shadow components.
Resume(MultiGather)
VoxData
Fetch
Resume(VoxData)
Memo(VoxModel)
Memo(VoxNode)
Memo(Primitive)
Memo(VoxLayer)
GeometryData
Data
Resume(VoxLayer)
ShaderLitMaterial
Memo(RawFaces)
Memo(RawFaces)
node_Node
GeometryData
Data
PBRMaterial
Memo(Mesh)
Memo(Primitive)
Memo(RawFaces)
Animate
Resume(Animate)
Memo(PointLight)
Memo(PointHelper)
Memo(PointLayer)
Memo(RawQuads)
Memo(PointLight)
Memo(PointHelper)
Memo(PointLayer)
Memo(RawQuads)
Show Ray Iterations
Previous
Next
Compose voxel raytracer with PBR pipeline


Raytrace a .vox model using the Voxel package. Composes with the existing lighting and shadow components.
Resume(MultiGather)
VoxData
Fetch
Resume(VoxData)
Memo(VoxModel)
Memo(VoxNode)
Memo(Primitive)
Memo(VoxLayer)
GeometryData
Data
Resume(VoxLayer)
ShaderLitMaterial
Memo(RawFaces)
Memo(RawFaces)
node_Node
GeometryData
Data
PBRMaterial
Memo(Mesh)
Memo(Primitive)
Memo(RawFaces)
Animate
Resume(Animate)
Memo(PointLight)
Memo(PointHelper)
Memo(PointLayer)
Memo(RawQuads)
Memo(PointLight)
Memo(PointHelper)
Memo(PointLayer)
Memo(RawQuads)
Show Ray Iterations
Previous
Next
Compose voxel raytracer with PBR pipeline

Raytrace a .vox model using the Voxel package. Composes with the existing lighting and shadow components.

Props | Fiber | Vertex | Fragment | Geometry | **Targets**

Fiber
ShadedRender

Fiber
ShadedRender
Fiber
Fiber
Fiber

Resume(MultiGather)
Memo(ShadowPass)
Memo(ShadowOmniPass)
Memo(ShadowOmniPass)
Memo(ShadowOmniPass)
Memo(ColorPass)

emo(Pass)
lemo(FullScreenRenderer)
econcile

Memo(RawFullScreen)

oot(PassReconciler)

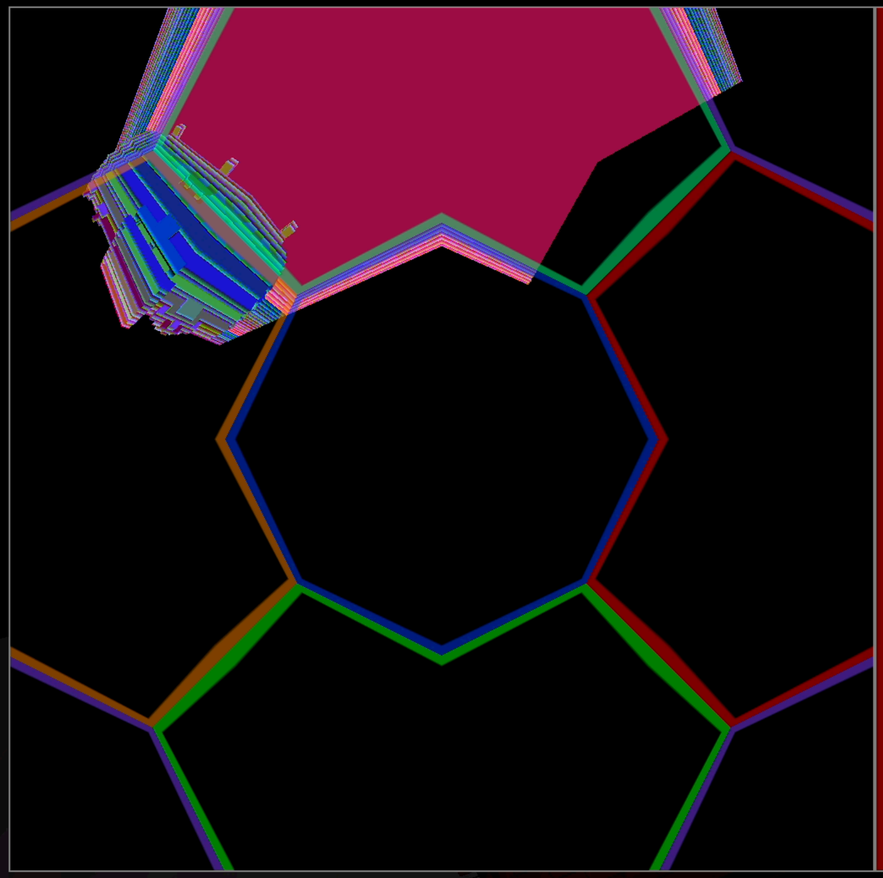
Fiber

Resume(MultiGather)
Memo(ColorPass)

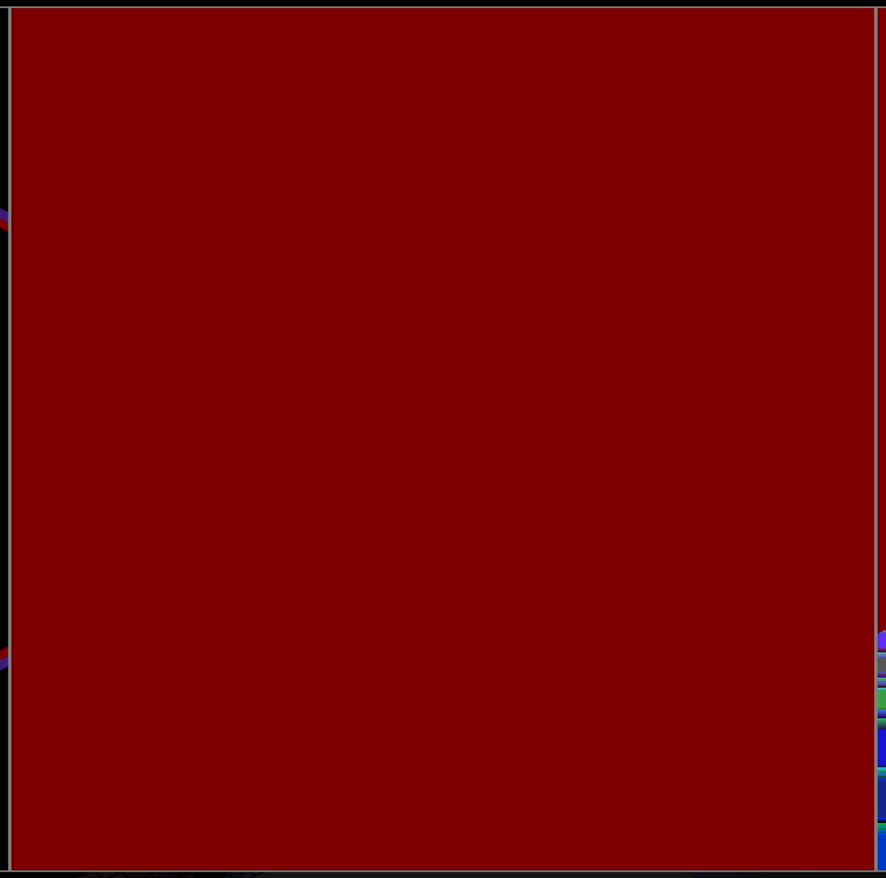
L
es)

orState)

er)



texture_depth_cube – depth32float
<ShadowOmniPass>



texture_depth_cube (face 1) – depth32float
<ShadowOmniPass>

texture_depth_cu
<ShadowOmniPass

Show Ray Iterations

◀ Previous | Next ▶ | < > Code | Geometry - Voxels

Raytrace a .vox model using the Voxel package. Composes with the existing lighting and shadow components.

Props | Fiber | Vertex | Fragment | Geometry | **Targets**

Fiber
ShadedRender

Fiber
ShadedRender
Fiber
Fiber
Fiber

Resume(MultiGather)
Memo(ShadowPass)
Memo(ShadowOmniPass)
Memo(ShadowOmniPass)
Memo(ShadowOmniPass)
Memo(ColorPass)

emo(Pass)
lemo(FullScreenRenderer)
econcile

Memo(RawFullScreen)

oot(PassReconciler)

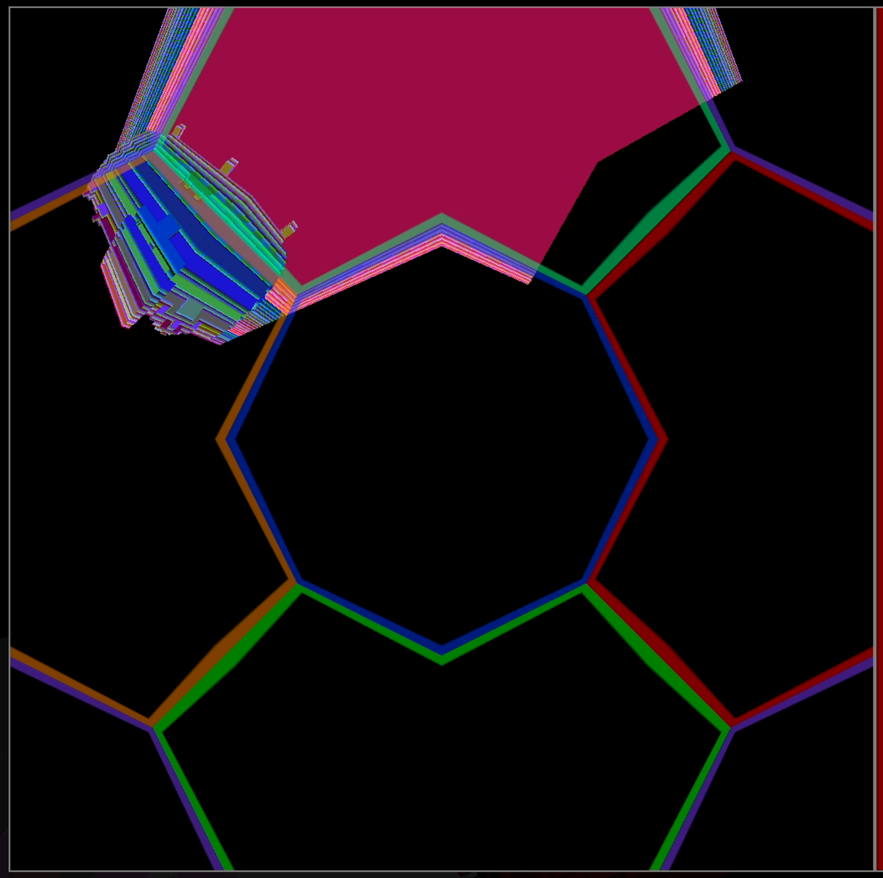
Fiber

Resume(MultiGather)
Memo(ColorPass)

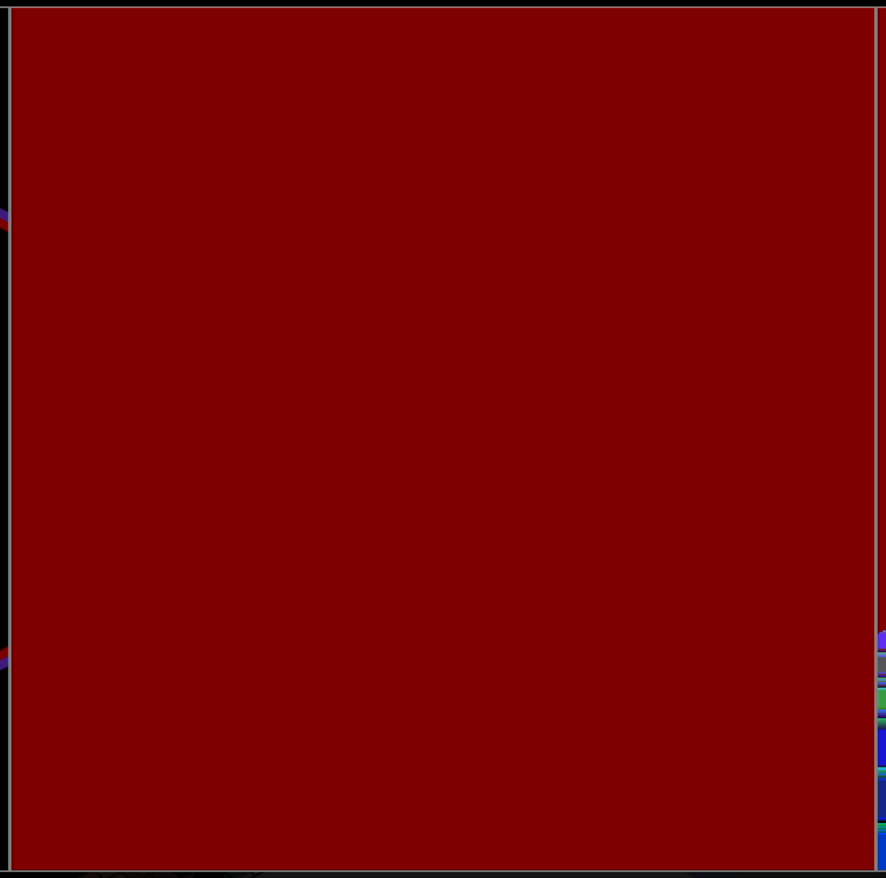
L
es)

orState)

er)



texture_depth_cube – depth32float
<ShadowOmniPass>

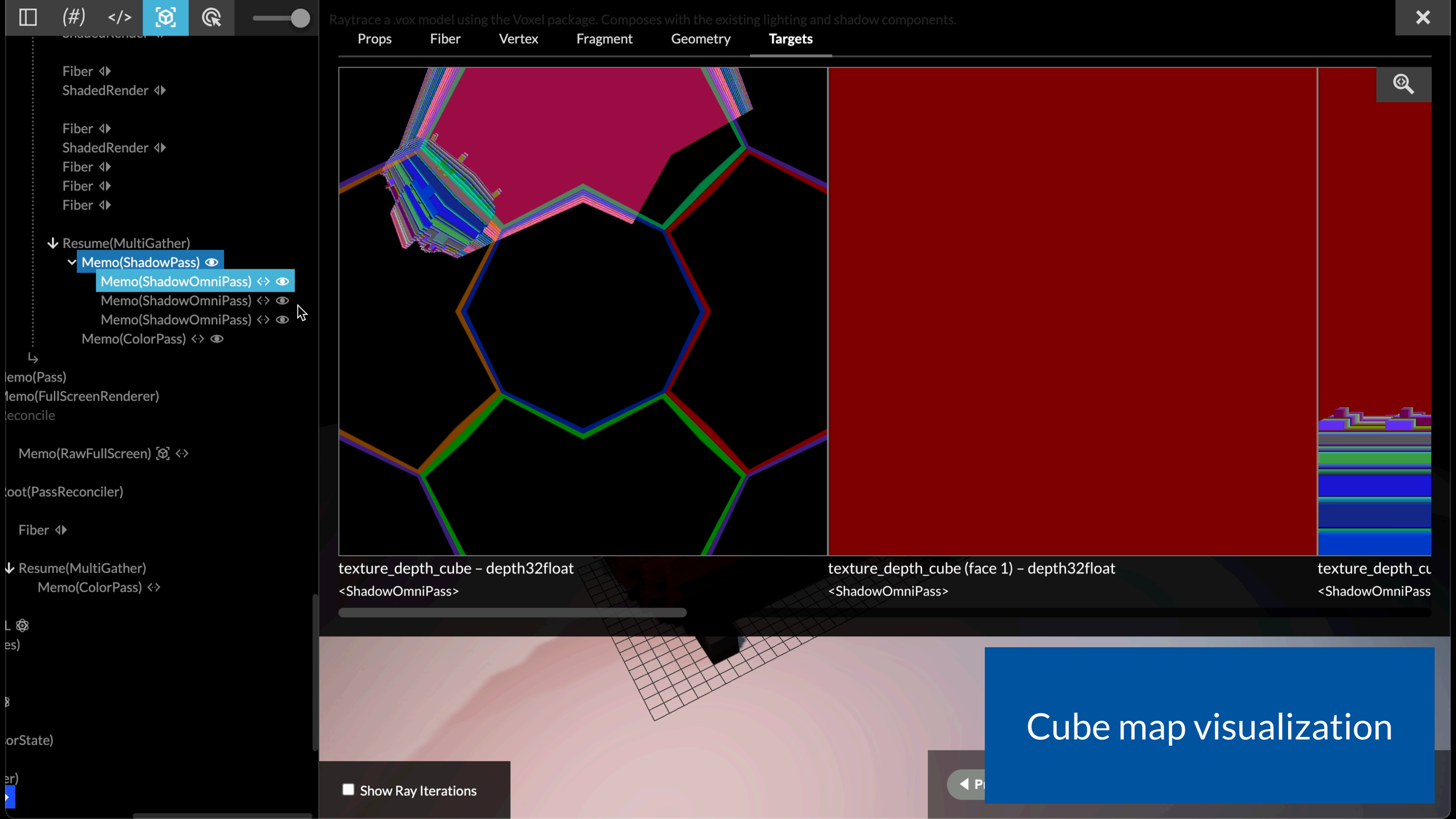


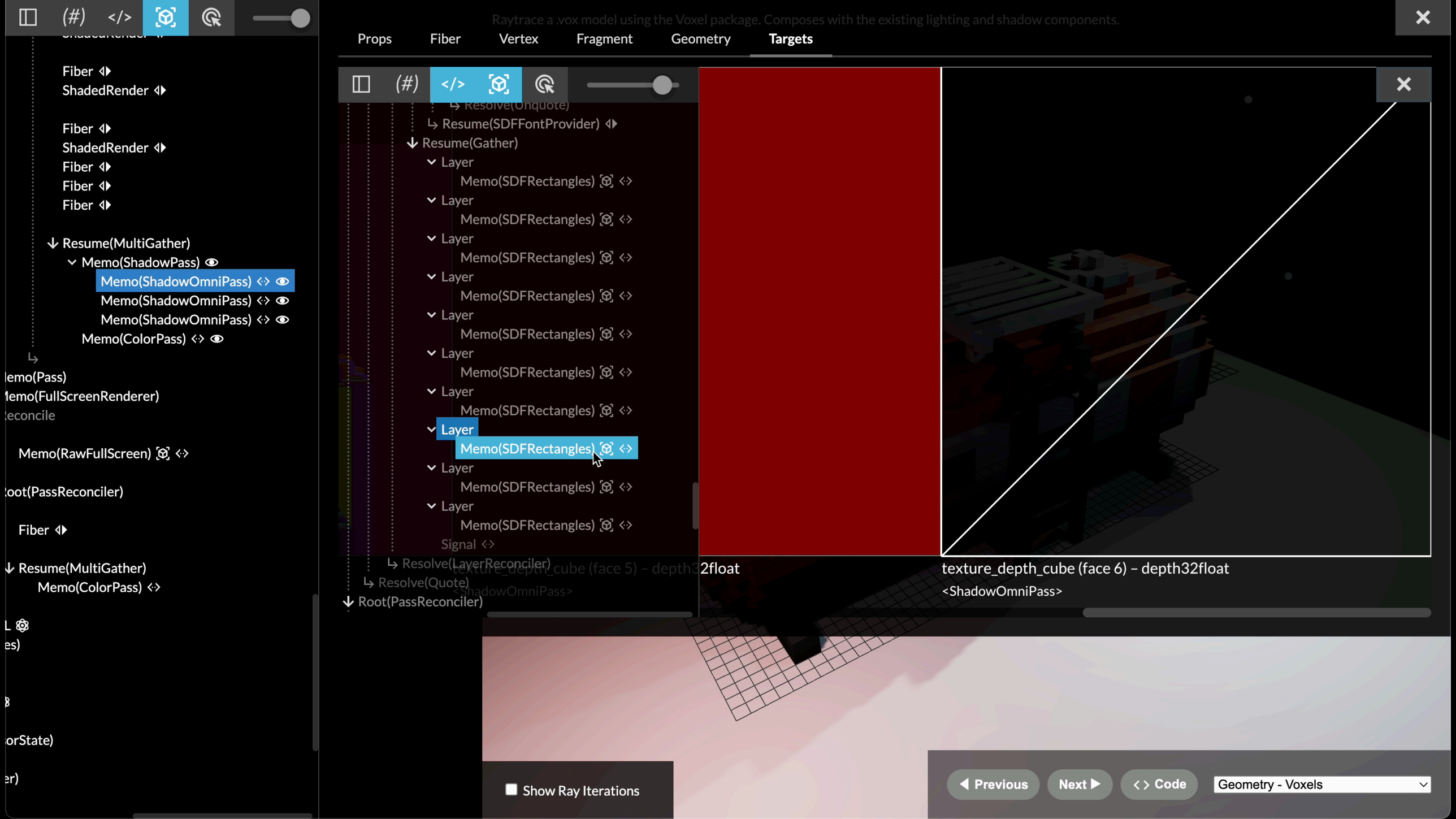
texture_depth_cube (face 1) – depth32float
<ShadowOmniPass>

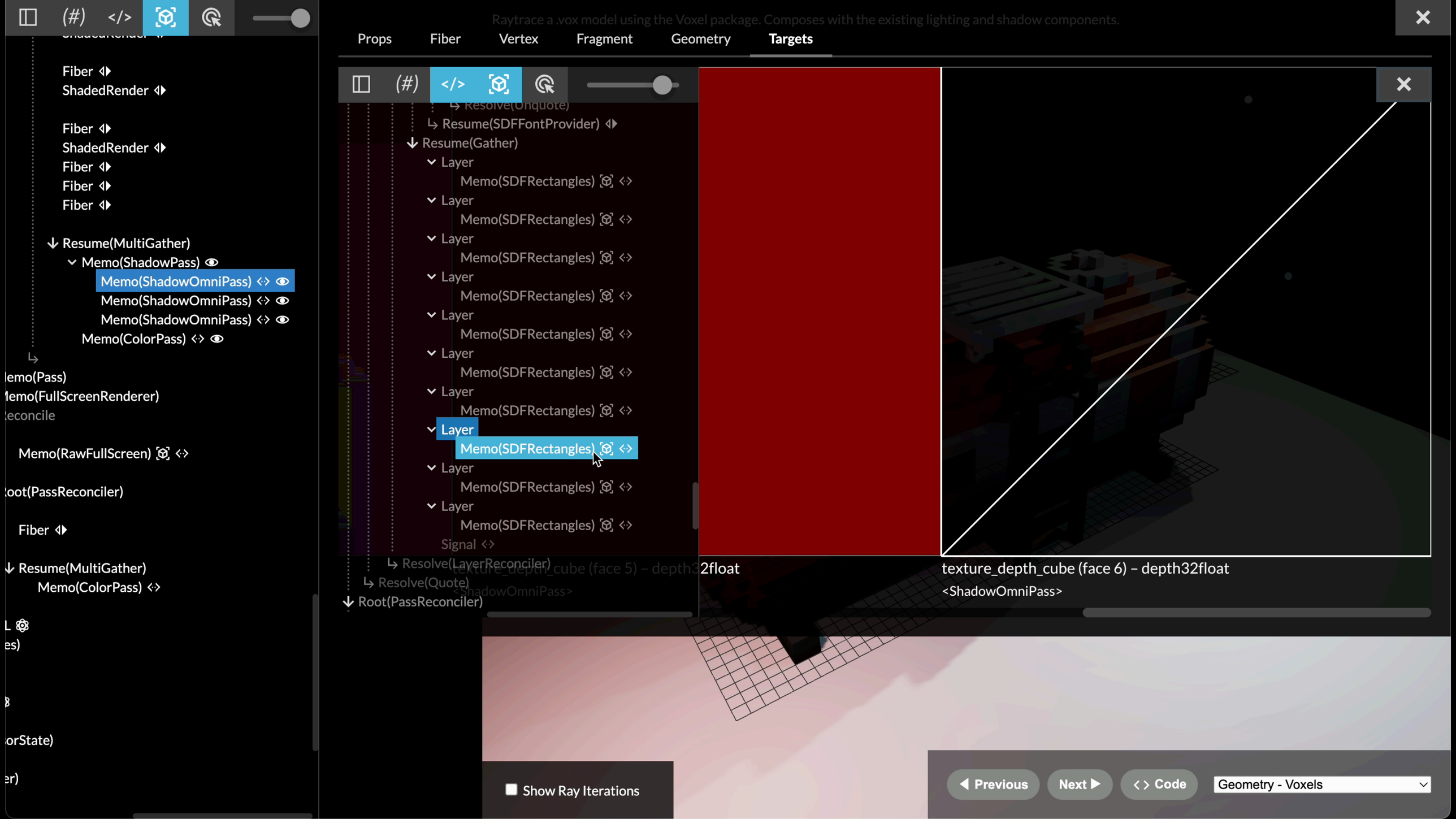
texture_depth_cu
<ShadowOmniPass

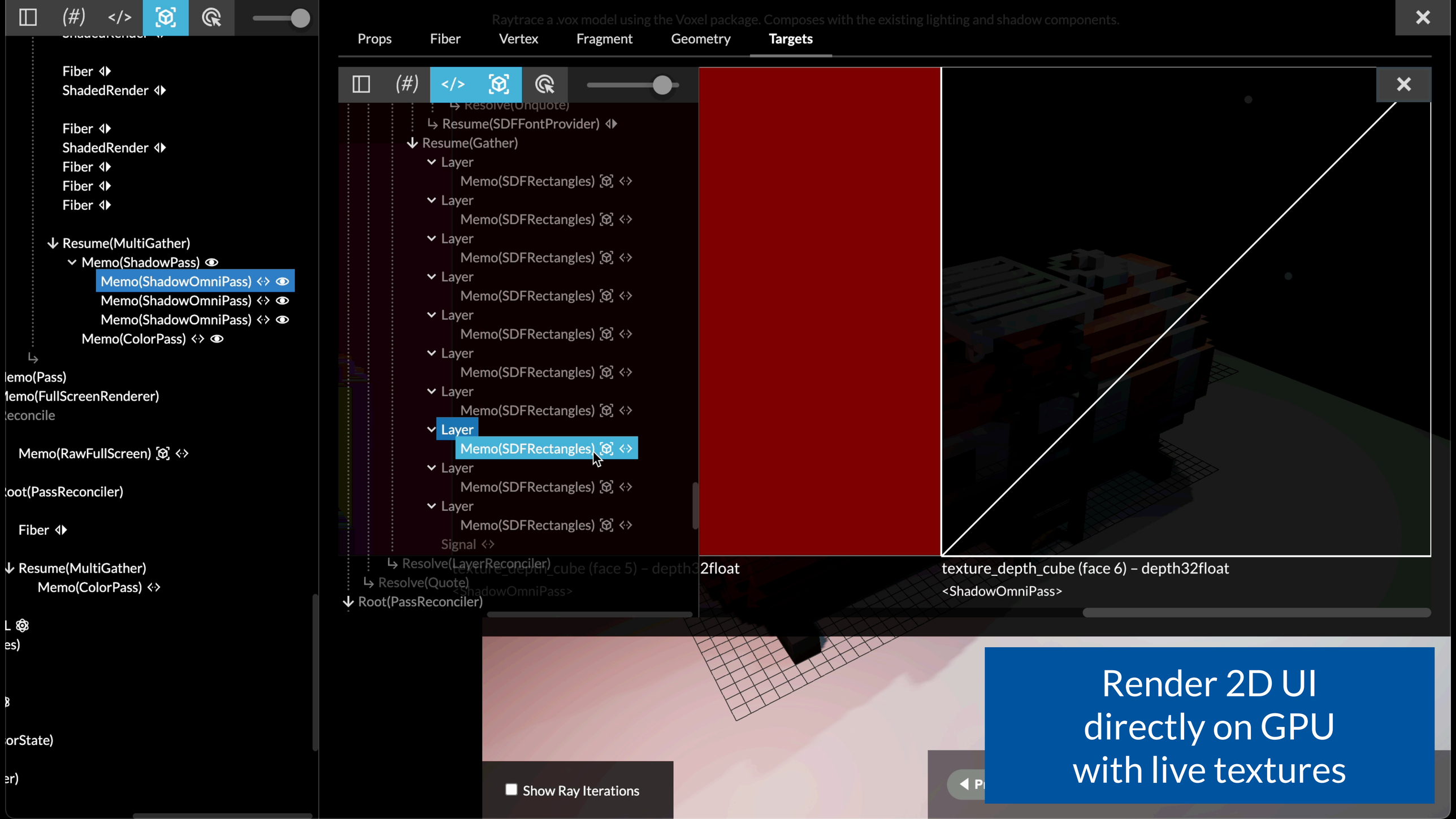
Show Ray Iterations

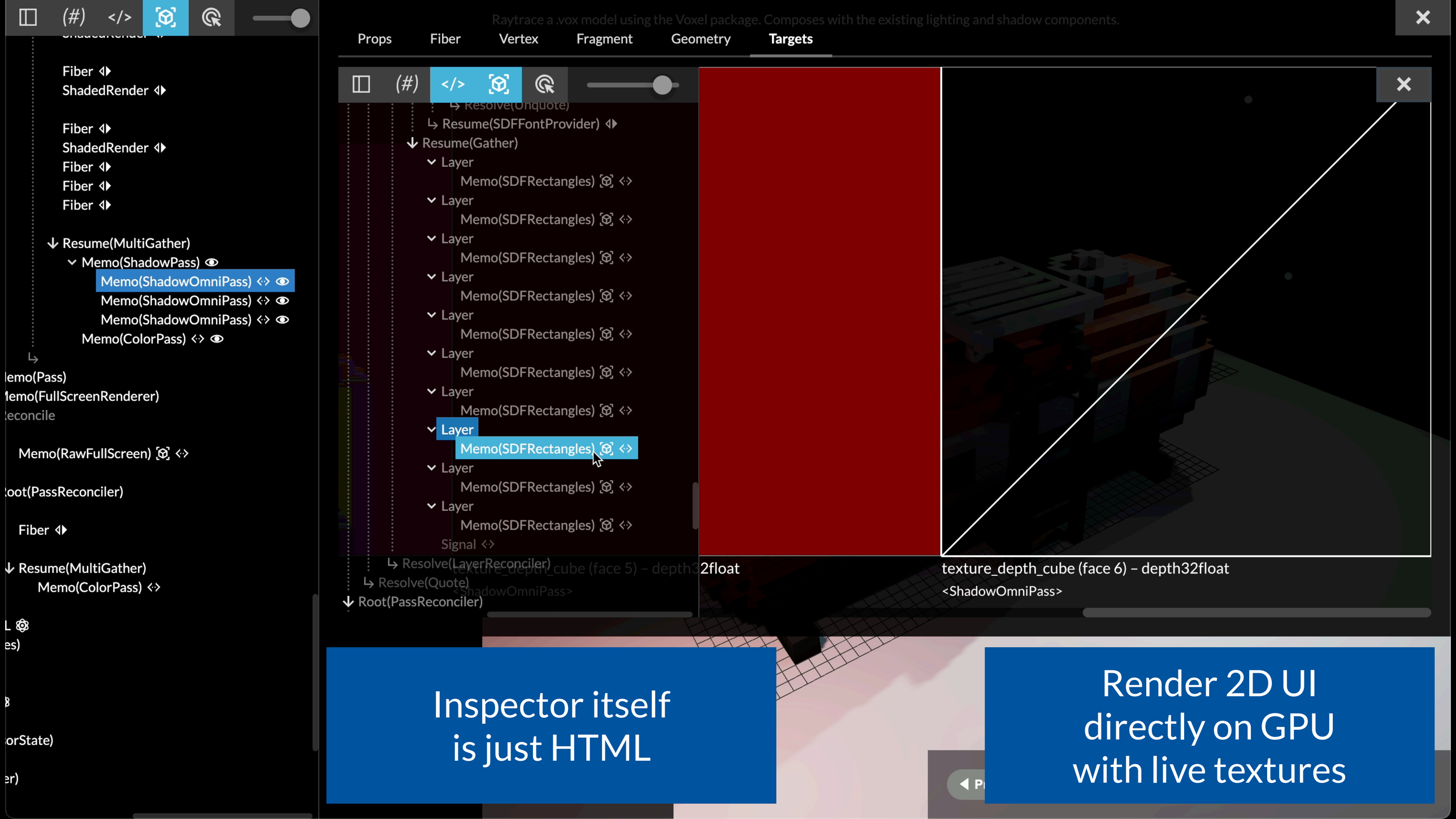
◀ Previous | Next ▶ | < > Code | Geometry - Voxels


Raytrace a .vox model using the Voxel package. Composes with the existing lighting and shadow components.
Props
Fiber
Vertex
Fragment
Geometry
Targets
Fiber
ShadedRender
Fiber
ShadedRender
Fiber
Fiber
Fiber
Resume(MultiGather)
Memo(ShadowPass)
Memo(ShadowOmniPass)
Memo(ShadowOmniPass)
Memo(ShadowOmniPass)
Memo(ColorPass)
Memo(RawFullScreen)
Fiber
Resume(MultiGather)
Memo(ColorPass)
texture_depth_cube – depth32float
<ShadowOmniPass>
texture_depth_cube (face 1) – depth32float
<ShadowOmniPass>
texture_depth_cu
<ShadowOmniPass
Show Ray Iterations
Cube map visualization


Raytrace a .vox model using the Voxel package. Composes with the existing lighting and shadow components.
Props
Fiber
Vertex
Fragment
Geometry
Targets
Fiber
ShadedRender
Fiber
ShadedRender
Fiber
Fiber
Fiber
Resume(MultiGather)
Memo(ShadowPass)
Memo(ShadowOmniPass)
Memo(ShadowOmniPass)
Memo(ShadowOmniPass)
Memo(ColorPass)
Memo(RawFullScreen)
Fiber
Resume(MultiGather)
Memo(ColorPass)
Resume(Unquote)
Resume(SDFFontProvider)
Resume(Gather)
Layer
Memo(SDFRectangles)
Signal
Resolve(LayerReconciler)
Resolve(Quote)
Root(PassReconciler)
2float
texture_depth_cube (face 6) – depth32float
<ShadowOmniPass>
Show Ray Iterations
Previous
Next
Code
Geometry - Voxels


Raytrace a .vox model using the Voxel package. Composes with the existing lighting and shadow components.
Props
Fiber
Vertex
Fragment
Geometry
Targets
Fiber
ShadedRender
Fiber
ShadedRender
Fiber
Fiber
Fiber
Resume(MultiGather)
Memo(ShadowPass)
Memo(ShadowOmniPass)
Memo(ShadowOmniPass)
Memo(ShadowOmniPass)
Memo(ColorPass)
Memo(RawFullScreen)
Fiber
Resume(MultiGather)
Memo(ColorPass)
Resume(Unquote)
Resume(SDFFontProvider)
Resume(Gather)
Layer
Memo(SDFRectangles)
Signal
Resolve(LayerReconciler)
Resolve(Quote)
Root(PassReconciler)
2float
texture_depth_cube (face 6) – depth32float
<ShadowOmniPass>
Show Ray Iterations
Previous
Next
Code
Geometry - Voxels


Raytrace a .vox model using the Voxel package. Composes with the existing lighting and shadow components.
Props
Fiber
Vertex
Fragment
Geometry
Targets
Fiber
ShadedRender
Fiber
ShadedRender
Fiber
Fiber
Fiber
Resume(MultiGather)
Memo(ShadowPass)
Memo(ShadowOmniPass)
Memo(ShadowOmniPass)
Memo(ShadowOmniPass)
Memo(ColorPass)
Memo(RawFullScreen)
Fiber
Resume(MultiGather)
Memo(ColorPass)
Resume(Unquote)
Resume(SDFFontProvider)
Resume(Gather)
Layer
Memo(SDFRectangles)
Signal
Resolve(LayerReconciler)
Resolve(Quote)
Root(PassReconciler)
texture_depth_cube (face 5) – depth32float
<ShadowOmniPass>
texture_depth_cube (face 6) – depth32float
<ShadowOmniPass>
Show Ray Iterations
Render 2D UI
directly on GPU
with live textures


Raytrace a .vox model using the Voxel package. Composes with the existing lighting and shadow components.
Props
Fiber
Vertex
Fragment
Geometry
Targets
Fiber
ShadedRender
Fiber
ShadedRender
Fiber
Fiber
Fiber
Resume(MultiGather)
Memo(ShadowPass)
Memo(ShadowOmniPass)
Memo(ShadowOmniPass)
Memo(ShadowOmniPass)
Memo(ColorPass)
Memo(RawFullScreen)
Fiber
Resume(MultiGather)
Memo(ColorPass)
Resume(Unquote)
Resume(SDFFontProvider)
Resume(Gather)
Layer
Memo(SDFRectangles)
Signal
Resolve(LayerReconciler)
Resolve(Quote)
Root(PassReconciler)
texture_depth_cube (face 5) – depth32float
<ShadowOmniPass>
texture_depth_cube (face 6) – depth32float
<ShadowOmniPass>
Inspector itself is just HTML
Render 2D UI directly on GPU with live textures

Trouble in paradise

GPU Dispatch

GPU Dispatch

```
fn ( a, b, c, d )
```

GPU Dispatch

- Allocate GPU memory

```
fn ( a, b, c, d )
```

GPU Dispatch

- Allocate GPU memory
- Upload data

```
fn ( a, b, c, d )
```

GPU Dispatch

- Allocate GPU memory
- Upload data
- Compile shader

```
fn ( a, b, c, d )
```

GPU Dispatch

- Allocate GPU memory
- Upload data
- Compile shader

```
fn ( a, b, c, d )
```

GPU Dispatch

- Allocate GPU memory
- Upload data
- Compile shader

- Dispatch

```
fn ( a, b, c, d )
```

GPU Dispatch

- Allocate GPU memory
- Upload data
- Compile shader

- Dispatch
- Await

```
fn ( a, b, c, d )
```

GPU Dispatch

- Allocate GPU memory
- Upload data
- Compile shader

- Dispatch
- Await

```
fn ( a, b, c, d )
```

Every time you **split** a dispatch **into 2**,
you need **more glue**.

Shader Dispatch

fn

a

b

c

d

Threads (N > 1,000,000)

0	1	2	3	4	5	6	7	8	9	…
3	4	1	3	2	5	7	8	2	3	
XYZ	**XYZ**	**XYZ**	**XYZ**	**XYZ**	**XYZ**	**XYZ**	**XYZ**	**XYZ**	**XYZ**	

Shader Dispatch

fn(*a*, *b*, *c*, *d*)

Threads (N > 1,000,000)

0	1	2	3	4	5	6	7	8	9	…
3	4	1	3	2	5	7	8	2	3	
XYZ	**XYZ**	**XYZ**	**XYZ**	**XYZ**	**XYZ**	**XYZ**	**XYZ**	**XYZ**	**XYZ**	

Memory bandwidth is the **main bottleneck**

Spread Policies

fn

Threads (N > 1,000,000)

	0	1	2	3	4	5	6	7	8	9	…
1 to 1	Vertex					`array[i]`					
Repeat	Instanced					`array[i % n]`				`[i / n]`	
Lookup	Indexed					`array[lookup[i]]`					
Constant	Uniform					`array[0]`					

Spread Policies

+ Storage	`bigArray[randomAccess]`

fn

Threads (N > 1,000,000)

	0	1	2	3	4	5	6	7	8	9	…
1 to 1				Vertex		`array[i]`					
Repeat				Instanced		`array[i % n]`			`[i / n]`		
Lookup				Indexed		`array[lookup[i]]`					
Constant				Uniform		`array[0]`					

Spread Policies

+ Storage	`bigArray[randomAccess]`

fn

Threads (N > 1,000,000)

	0	1	2	3	4	5	6	
1 to 1				Vertex		`array[i]`		
Repeat				Instanced		`array[i % n]`		`[i / n]`
Lookup				Indexed		`array[lookup[i]]`		
Constant				Uniform		`array[0]`		

⚠ **Legacy Cruft**

Spread Policies

+ Storage `bigArray[randomAccess]`

fn

Threads (N > 1,000,000)

	0	1	2	3	4	5	6
1 to 1				Vertex		`array[i]`	
Repeat				Instanced		`array[i % n]`	`[i / n]`
Lookup				Indexed		`array[lookup[i]]`	
Constant				Uniform		`array[0]`	

⚠ **Legacy Cruft**

Completely different **types and APIs** for each on both CPU and GPU 🤬

Spread Policies

+ Storage `bigArray[randomAccess]`

+ Restrictions on control flow and grouping 🫠

fn

Threads (N > 1,000,000)

	0	1	2	3	4	5	6
1 to 1				Vertex		`array[i]`	
Repeat				Instanced		`array[i % n]`	`[i / n]`
Lookup				Indexed		`array[lookup[i]]`	
Constant				Uniform		`array[0]`	

⚠ **Legacy Cruft**

Completely different **types and APIs** for each on both CPU and GPU 🤬

Spread Policies

Problem #1

\+ Storage `bigArray[randomAccess]`

\+ Restrictions on control flow and grouping 🫠

fn

Threads (N > 1,000,000)

	0	1	2	3	4	5	6
1 to 1				Vertex		`array[i]`	
Repeat				Instanced		`array[i % n]`	`[i / n]`
Lookup				Indexed		`array[lookup[i]]`	
Constant				Uniform		`array[0]`	

⚠ **Legacy Cruft**

<u>Completely</u> different **types and APIs** for each on both CPU and GPU 🤬

Spread Policies

Problem #1

+ Storage `bigArray[randomAccess]`
+ Restrictions on control flow and grouping 🫠

fn

Threads (N > 1,000,000)

	0	1	2	3	4	5	6	7	8	9	…
1 to 1				Vertex		`array[i]`					
Repeat				Instanced		`array[i % n]`		`[i / n]`			
Lookup				Indexed		`array[lookup[i]]`					
Constant				Uniform		`array[0]`					

Completely different **types and APIs** for each on both CPU and GPU 🤬

Spread Policies

Problem #1

+ Storage `bigArray[randomAccess]`
+ Restrictions on control flow and grouping 🤐

fn

Threads (N > 1,000,000)

	5	6	7	8	9	…
1 to 1		`array[i]`				
Repeat		`array[i % n]`				
Lookup	Indexed	`array[lookup[i]]`				
Constant	Uniform	`array[0]`				

Batch <u>everything</u>

<u>Completely</u> different **types and APIs** for each on both CPU and GPU 🤬

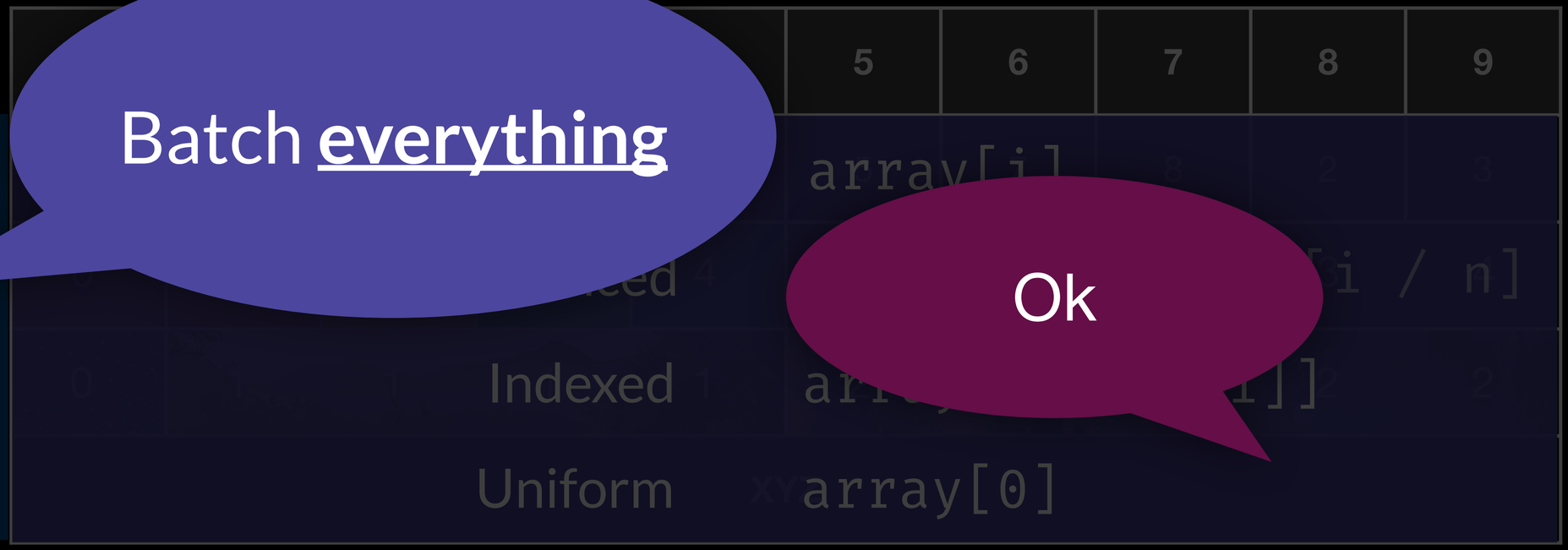
Spread Policies

Problem #1

+ Storage `bigArray[randomAccess]`
+ Restrictions on control flow and grouping 🤐

fn

Threads (N > 1,000,000)



Batch **everything**

Ok

Completely different **types and APIs** for each on both CPU and GPU 🤬

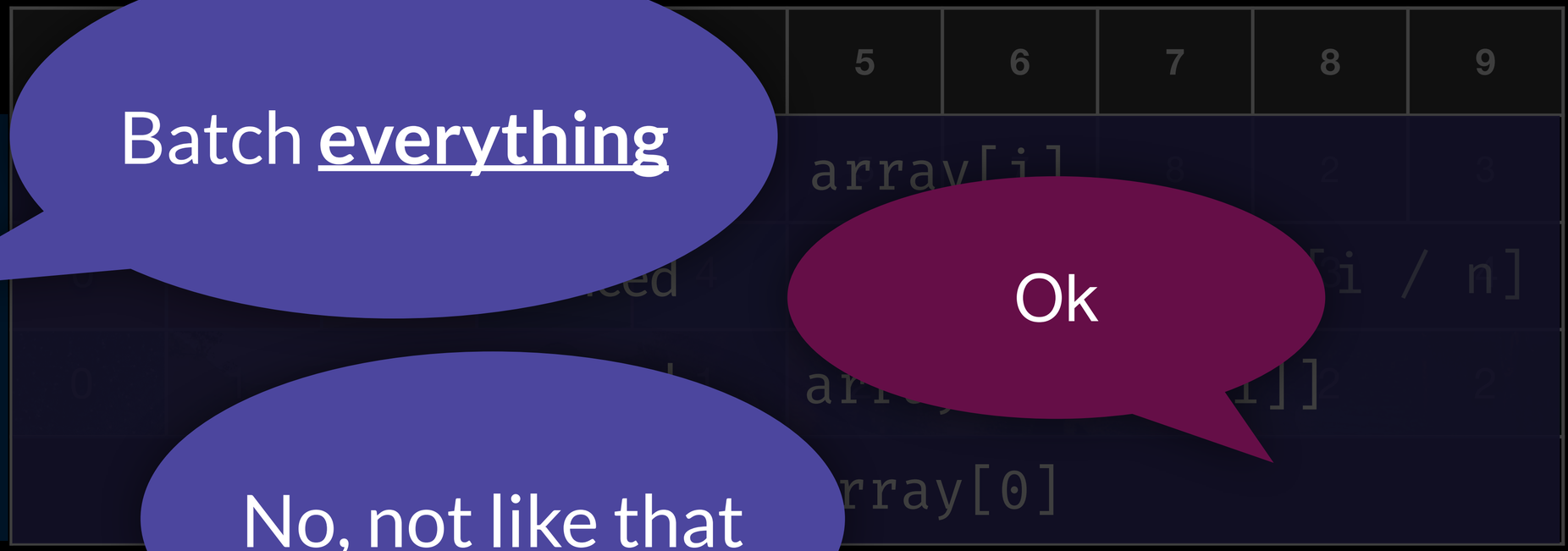
Spread Policies

Problem #1

+ Storage `bigArray[randomAccess]`
+ Restrictions on control flow and grouping 🤐

fn

Threads (N > 1,000,000)



1 to 1

Repeat

Lookup

Constant

Batch **everything**

Ok

No, not like that

Completely diff **is** for each on both CPU and GPU 🤬

Pipeline Dispatch

Pipeline Dispatch

```
fn ( input: Texture, output: Texture )
```

Pipeline Dispatch

```
fn ( input: Texture, output: Texture )
```

Map **every pixel** 1-to-1

Pipeline Dispatch

```
fn ( input: Texture, output: Texture )
```

Map **every pixel** 1-to-1

No spread

Binding #0

Binding #1

Render Pass

Resource Binding

Sampler Binding

Color Attachme

```
fn ( input: Texture, vertices: T[ ], output: Target )
```

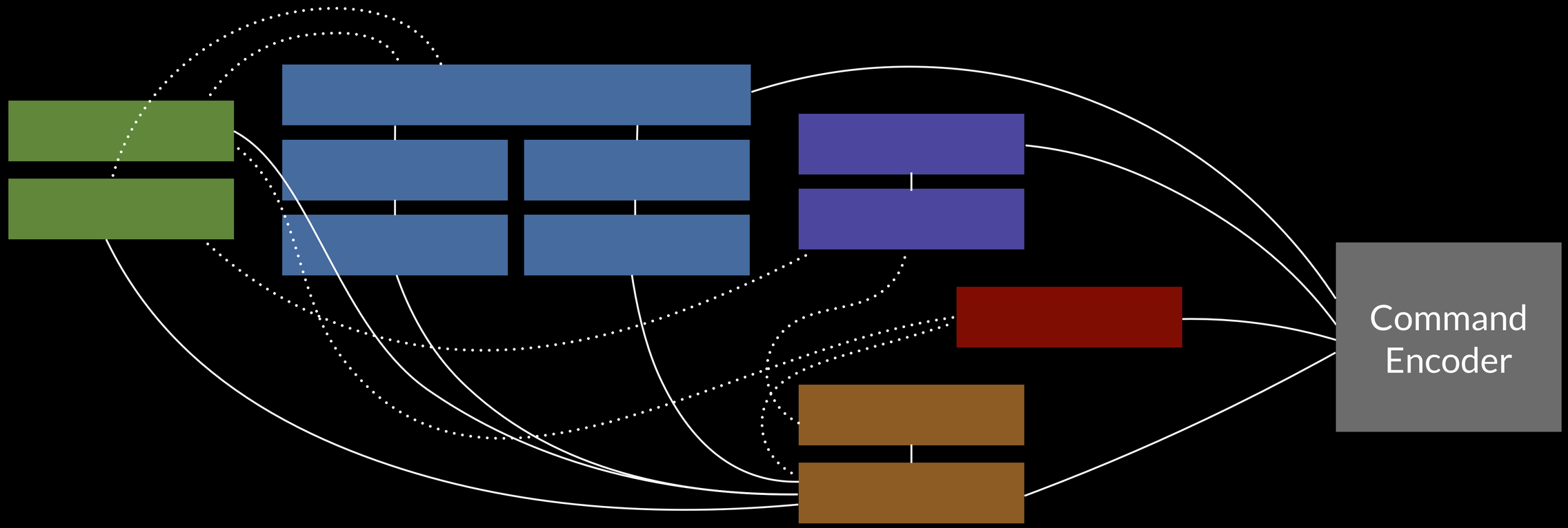
Vertex Shader

Vertex Buffer

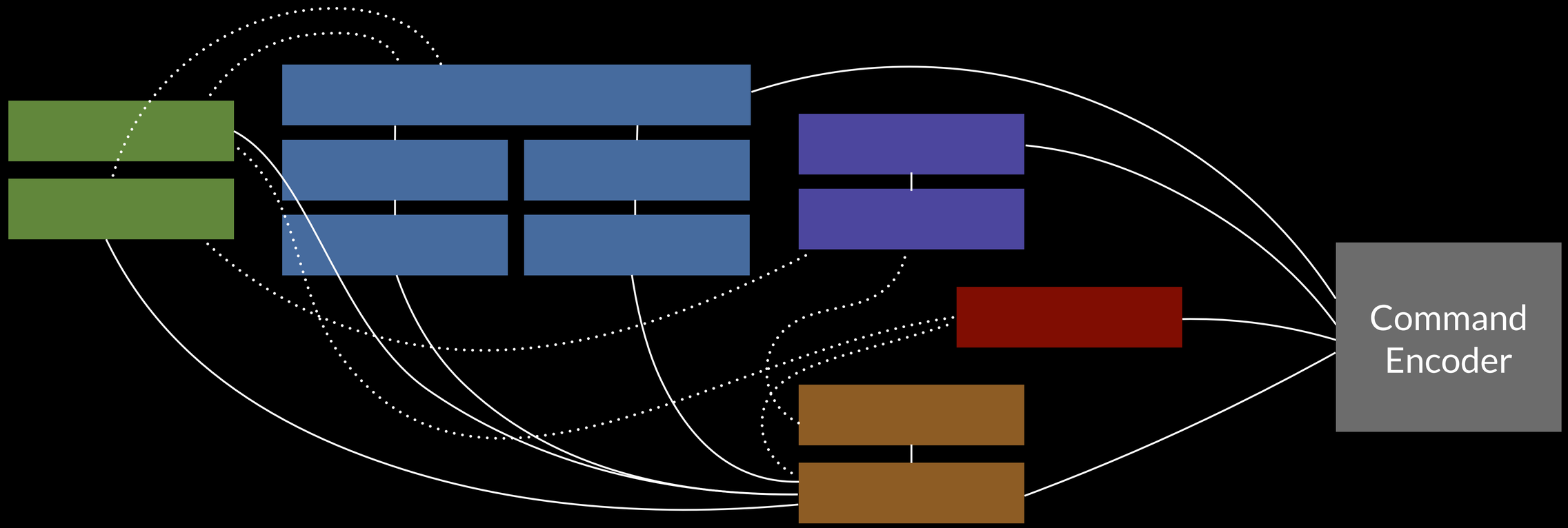
Color State

Fragment Shader

Pipeline

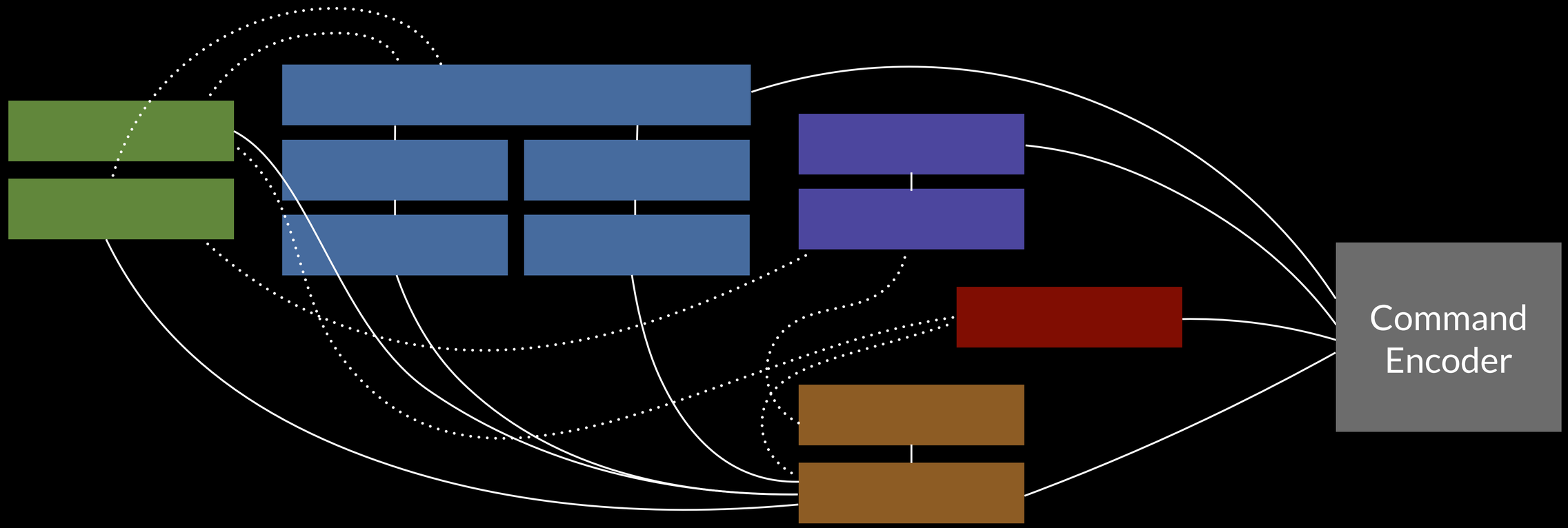


*(**S**hader<**T**, **U**, **V**>) => (Pipeline<**S**, **T**, **U**, **V**>)*
*=> (**T**argets, **U**niforms, **V**ertices)*



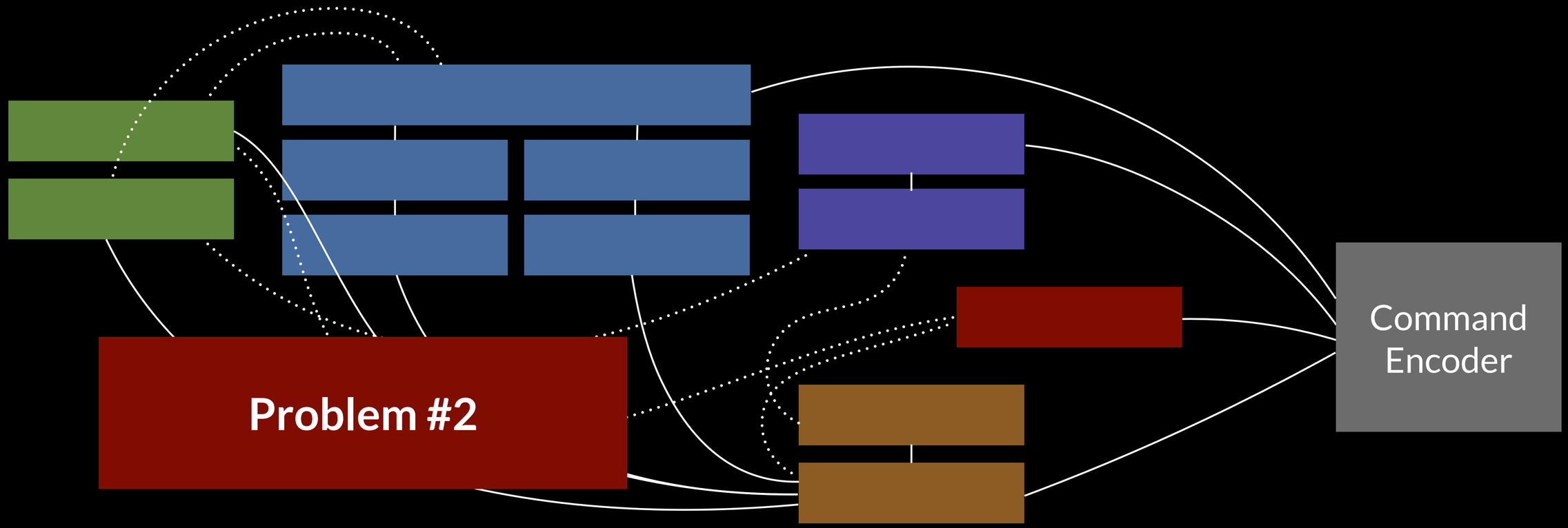
*(**S**hader<**T**, **U**, **V**>) => (Pipeline<**S**, **T**, **U**, **V**>)*
*=> (**T**argets, **U**niforms, **V**ertices)*

The calling convention is
a big nested data structure



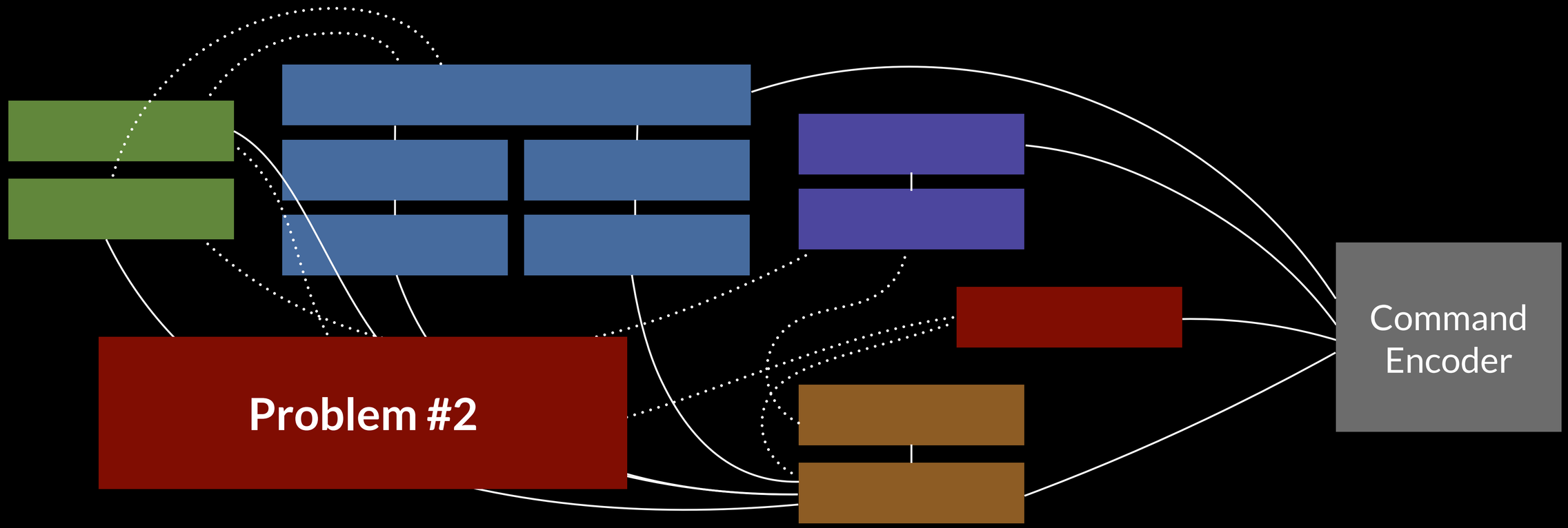
*(Shader<**T**, **U**, **V**>) ↻ (Pipeline<**S**, **T**, **U**, **V**>)*
↻ (Targets ↻ Uniforms ↻ Vertices)

The calling convention is
a big nested data structure



*(Shader<**T**, **U**, **V**>) ↻ (Pipeline<**S**, **T**, **U**, **V**>)*
↻ (Targets ↻ Uniforms ↻ Vertices)

The calling convention is
a big nested data structure

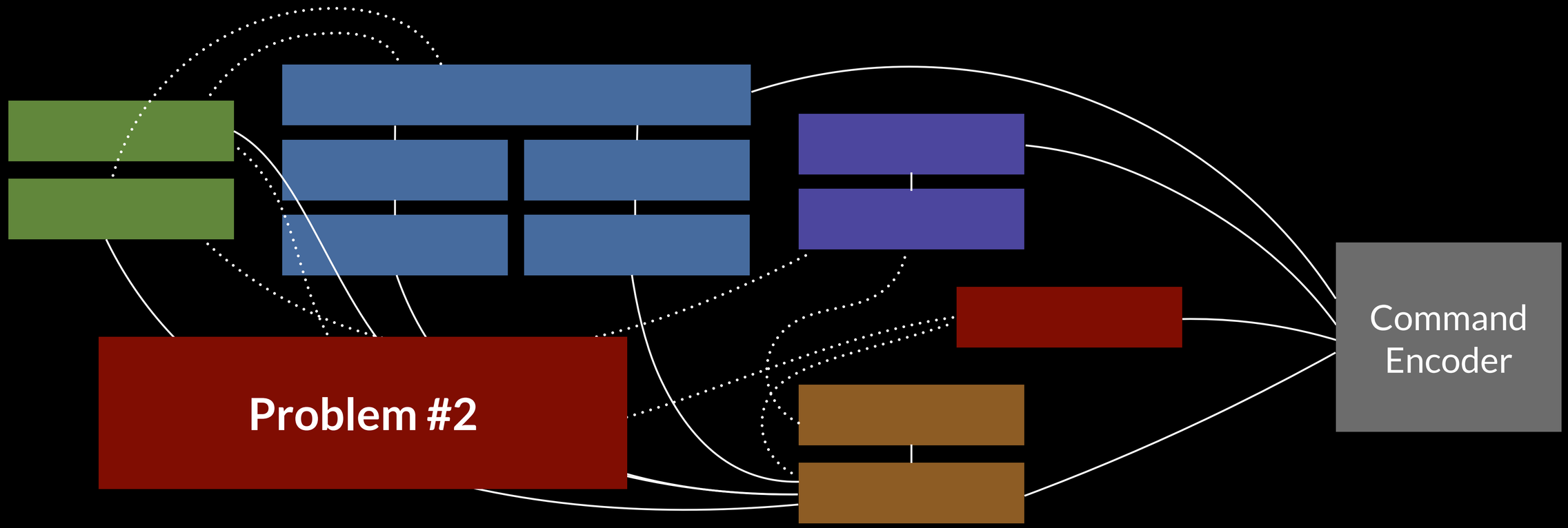


Problem #2

*(Shader<**T, U, V**>) & (Pipeline<**S, T, U, V**>)*
& (Targets & Uniforms & Vertices)

Renderable *formats*
😡

The calling convention is
a big nested data structure



Problem #2

*(Shader<**T, U, V**>) & (Pipeline<**S, T, U, V**>)*
& (Targets & Uniforms & Vertices)

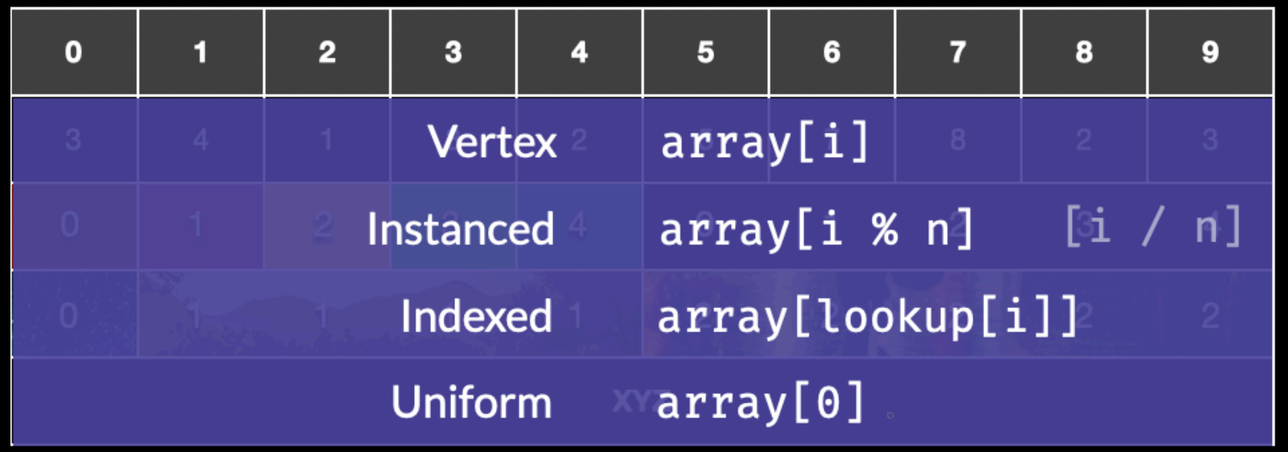
Renderable *formats*
😡

The calling convention is
a big nested data structure

Minimum
binding size 😭

Spread Policies

Problem #1



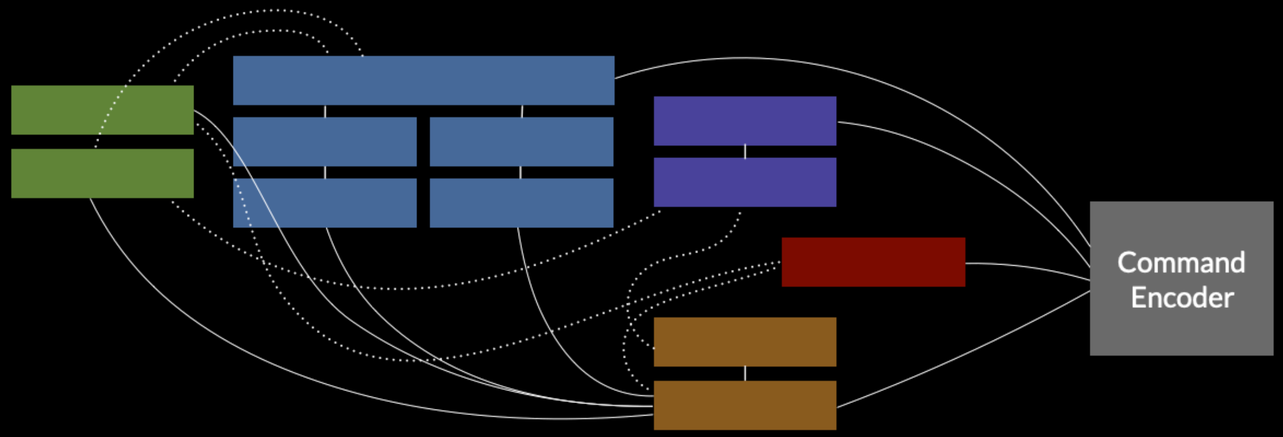
Spread Policies

Problem #1

0	1	2	3	4	5	6	7	8	9
Vertex					array[i]				
Instanced					array[i % n]			[i / n]	
Indexed					array[lookup[i]]				
Uniform					array[0]				

Pipeline Dispatch

Problem #2



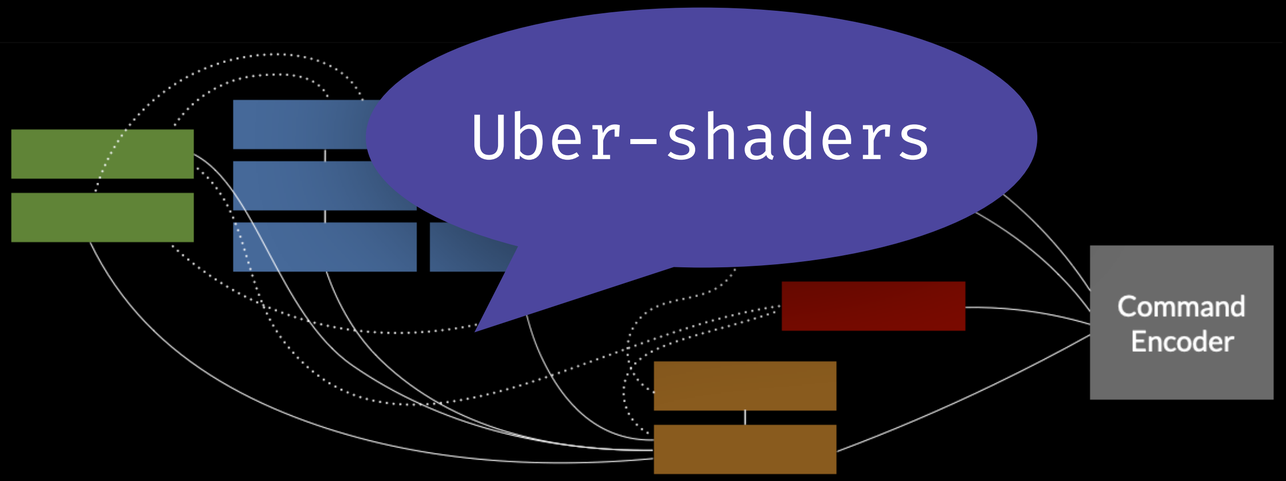
Spread Policies

Problem #1

0	1	2	3	4	5	6	7	8	9
Vertex					array[i]				
Instanced					array[i % n]			[i / n]	
Indexed					array[lookup[i]]				
Uniform					array[0]				

Pipeline Dispatch

Problem #2



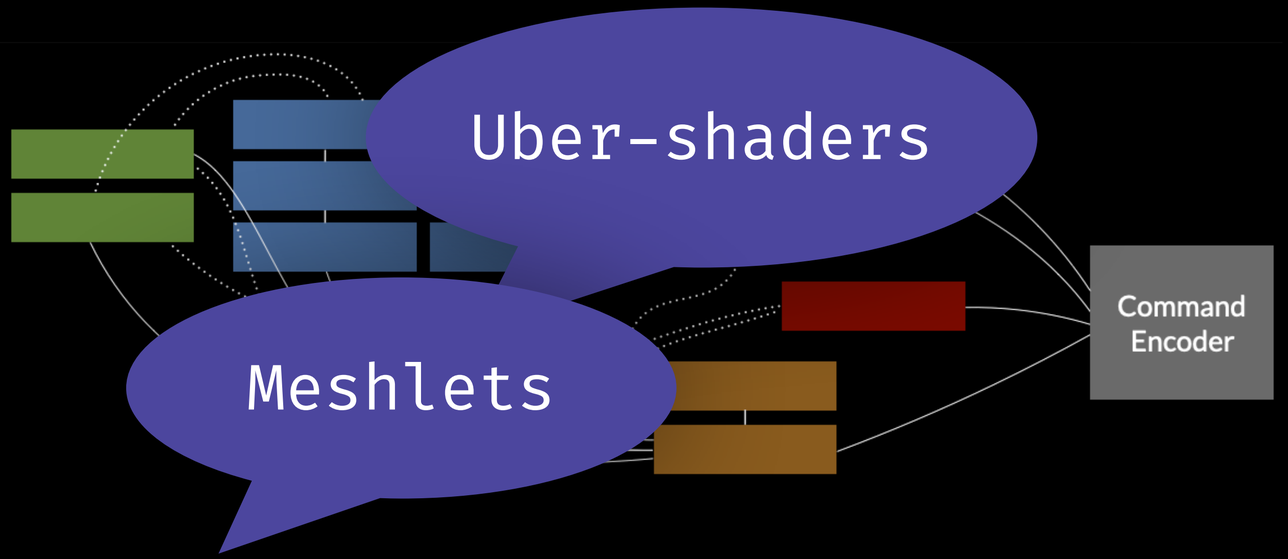
Spread Policies

Problem #1

0	1	2	3	4	5	6	7	8	9
Vertex					array[i]				
Instanced					array[i % n]			[i / n]	
Indexed					array[lookup[i]]				
Uniform					array[0]				

Pipeline Dispatch

Problem #2



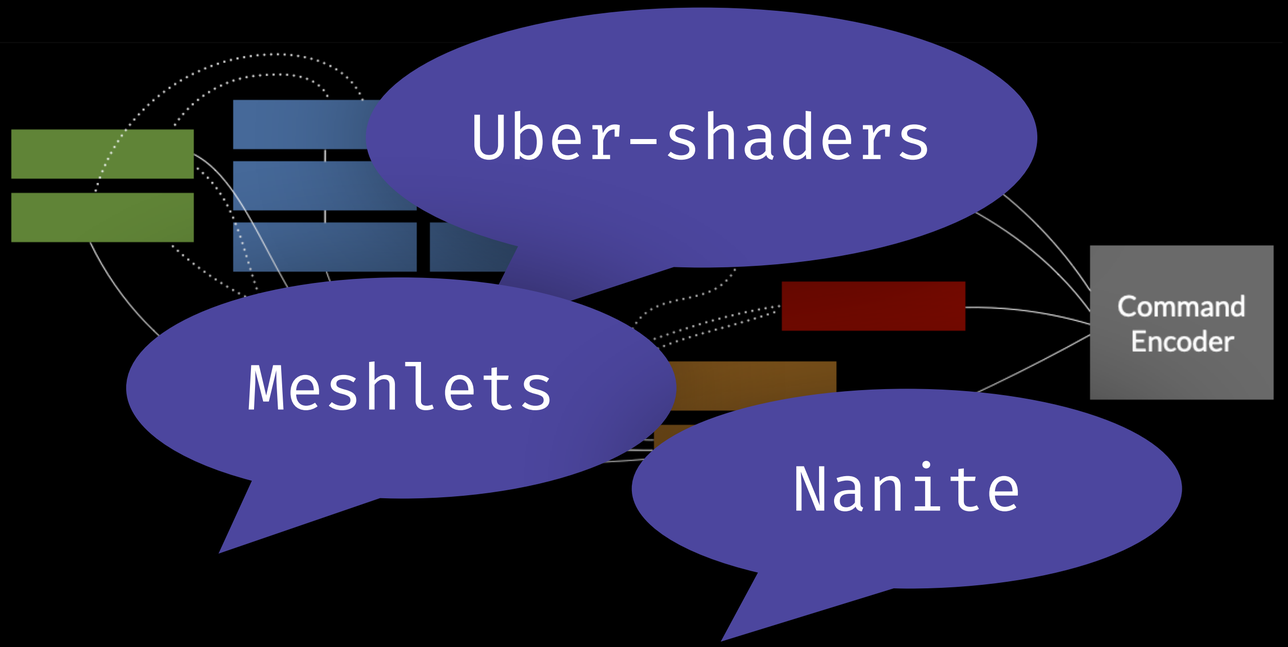
Spread Policies

Problem #1

0	1	2	3	4	5	6	7	8	9
Vertex					array[i]				
Instanced					array[i % n]			[i / n]	
Indexed					array[lookup[i]]				
Uniform					array[0]				

Pipeline Dispatch

Problem #2



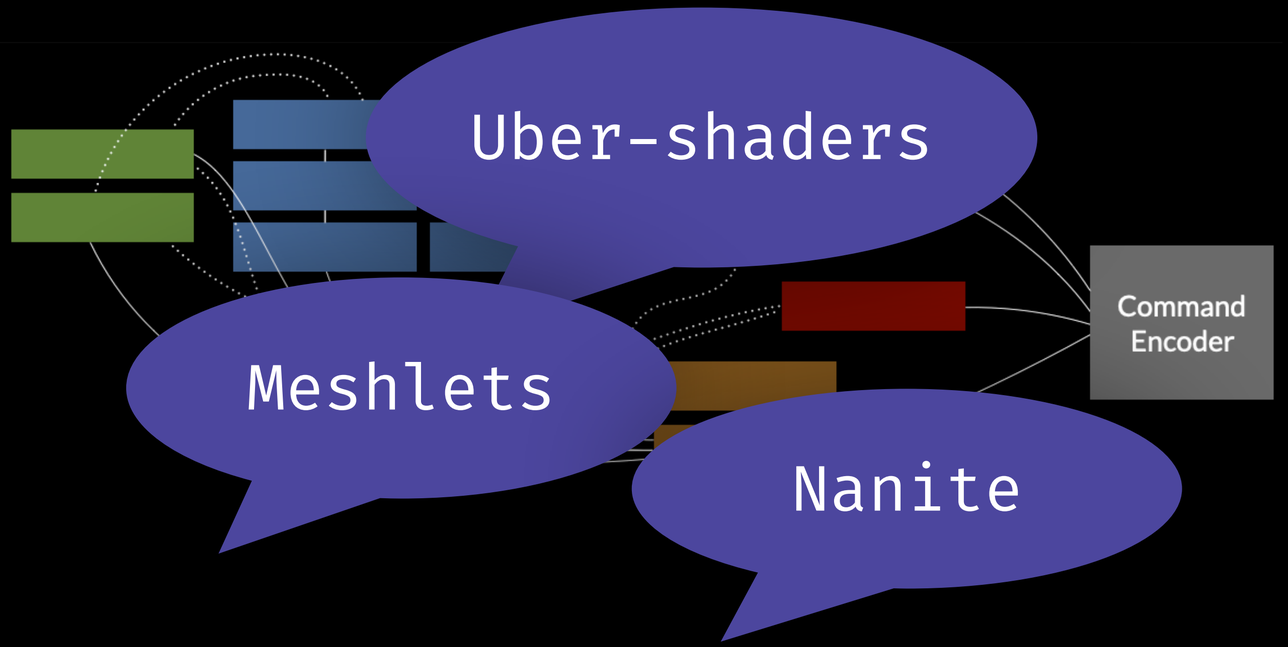
Spread Policies

We can fix this

0	1	2	3	4	5	6	7	8	9
Vertex					array[i]				
Instanced					array[i % n]			[i / n]	
Indexed					array[lookup[i]]				
Uniform					array[0]				

Pipeline Dispatch

Problem #2



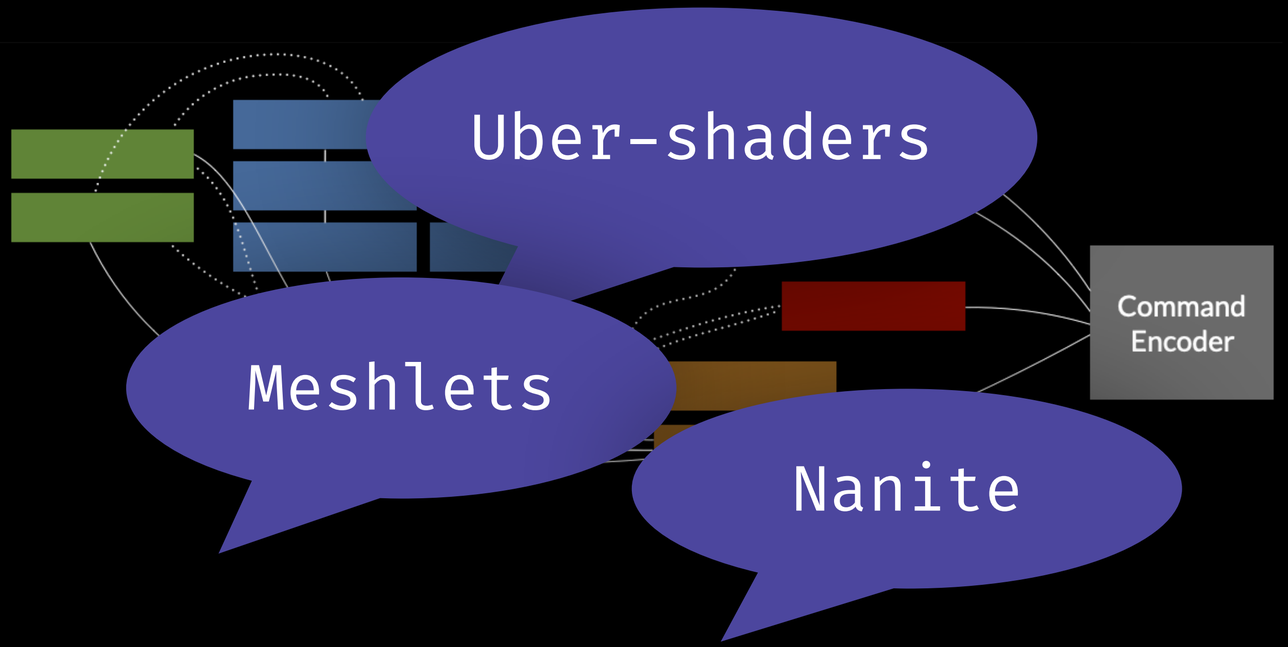
Spread Policies

We can fix this

0	1	2	3	4	5	6	7	8	9
Vertex					array[i]				
Instanced					array[i % n]			[i / n]	
Indexed					array[lookup[i]]				
Uniform					array[0]				

Pipeline Dispatch

We can't fix this



WGSL Extensions

"How do I compose shaders?"

@use-gpu/shader

"How do I compose shaders?"

@use-gpu/shader

"How do I compose shaders?"

Shader linker with
module system in WGSL

```
use '@use-gpu/wgsl/use/types'::{ SolidVertex };
use '@use-gpu/wgsl/geometry/quad'::{ getQuadIndex };
use '@use-gpu/wgsl/geometry/strip'::{ getStripIndex };
use '@use-gpu/wgsl/geometry/line'::{ getLineJoin };
```

@use-gpu/shader

"How do I compose shaders?"

Shader linker with **module system** in WGSL

Turn inputs into **getters**

```
use '@use-gpu/wgsl/use/types'::{ SolidVertex };
use '@use-gpu/wgsl/geometry/quad'::{ getQuadIndex };
use '@use-gpu/wgsl/geometry/strip'::{ getStripIndex };
use '@use-gpu/wgsl/geometry/line'::{ getLineJoin };
```

```
array[i]  ➡  @link fn getValue(i)
```

```
@optional @link fn getWidth(i: u32) -> f32 { return 1.0; };
```

@use-gpu/shader

"How do I compose shaders?"

Shader linker with **module system** in WGSL

```
use '@use-gpu/wgsl/use/types'::{ SolidVertex };
use '@use-gpu/wgsl/geometry/quad'::{ getQuadIndex };
use '@use-gpu/wgsl/geometry/strip'::{ getStripIndex };
use '@use-gpu/wgsl/geometry/line'::{ getLineJoin };
```

Turn inputs into **getters**

```
array[i]  ➡  @link fn getValue(i)
@optional @link fn getWidth(i: u32) -> f32 { return 1.0; };
```

Shader **imports** in TypeScript

```
import { getLineSegment } from '@use-gpu/wgsl/geometry/segment.wgsl';
import { getLineVertex } from '@use-gpu/wgsl/instance/vertex/line.wgsl';
```

@use-gpu/shader

"How do I compose shaders?"

Shader linker with **module system** in WGSL

```
use '@use-gpu/wgsl/use/types'::{ SolidVertex };
use '@use-gpu/wgsl/geometry/quad'::{ getQuadIndex };
use '@use-gpu/wgsl/geometry/strip'::{ getStripIndex };
use '@use-gpu/wgsl/geometry/line'::{ getLineJoin };
```

Turn inputs into **getters**

```
array[i]  ➡  @link fn getValue(i)
@optional @link fn getWidth(i: u32) -> f32 { return 1.0; };
```

Shader **imports** in TypeScript

```
import { getLineSegment } from '@use-gpu/wgsl/geometry/segment.wgsl';
import { getLineVertex } from '@use-gpu/wgsl/instance/vertex/line.wgsl';
```

Bind args by **handle**, **reference** or **lambda**

```
const getVertex = useShader(getLineVertex, [
  positions, scissor,
  uvs, sts,
  segments, colors, widths, depths, zBiases,
]);
```

@use-gpu/shader

"How do I compose shaders?"

Shader linker with **module system** in WGSL

```
use '@use-gpu/wgsl/use/types'::{ SolidVertex };
use '@use-gpu/wgsl/geometry/quad'::{ getQuadIndex };
use '@use-gpu/wgsl/geometry/strip'::{ getStripIndex };
use '@use-gpu/wgsl/geometry/line'::{ getLineJoin };
```

Turn inputs into **getters**

```
array[i]  ➡  @link fn getValue(i)
@optional @link fn getWidth(i: u32) -> f32 { return 1.0; };
```

Shader **imports** in TypeScript

```
import { getLineSegment } from '@use-gpu/wgsl/geometry/segment.wgsl';
import { getLineVertex } from '@use-gpu/wgsl/instance/vertex/line.wgsl';
```

Bind args by **handle**, **reference** or **lambda**

```
const getVertex = useShader(getLineVertex, [
  positions, scissor,
  uvs, sts,
  segments, colors, widths, depths, zBiases,
]);
```

Closures

@use-gpu/shader

"How do I compose shaders?"

Shader linker with **module system** in WGSL

```
use '@use-gpu/wgsl/use/types'::{ SolidVertex };
use '@use-gpu/wgsl/geometry/quad'::{ getQuadIndex };
use '@use-gpu/wgsl/geometry/strip'::{ getStripIndex };
use '@use-gpu/wgsl/geometry/line'::{ getLineJoin };
```

Turn inputs into **getters**

```
array[i]  ➡  @link fn getValue(i)

@optional @link fn getWidth(i: u32) -> f32 { return 1.0; };
```

Can also bind by name: **`bindBundle`**

Shader **imports** in TypeScript

```
import { getLineSegment } from '@use-gpu/wgsl/geometry/segment.wgsl';
import { getLineVertex } from '@use-gpu/wgsl/instance/vertex/line.wgsl';
```

Bind args by **handle**, **reference** or **lambda**

```
const getVertex = useShader(getLineVertex, [
  positions, scissor,
  uvs, sts,
  segments, colors, widths, depths, zBiases,
]);
```

Closures

Utility Module

```
fn signNotZero(xy: vec2<f32>) -> vec2<f32> {
  let s = sign(xy);
  return select(s, vec2<f32>(1.0, 1.0), s == vec2<f32>(0.0));
}

/** Assumes that v is a unit vector. The result is an octahedral vector on the [-1, +1] square. */
@export fn encodeOctahedral(v: vec3<f32>) -> vec2<f32> {
  let l1norm = abs(v.x) + abs(v.y) + abs(v.z);
  var result = v.xy * (1.0 / l1norm);
  if (v.z < 0.0) {
    result = (1.0 - abs(result.yx)) * signNotZero(result.xy);
  }
  return result;
}

/** Returns a unit vector. Argument o is an octahedral vector packed via encodeOctahedral,
    on the [-1, +1] square */
@export fn decodeOctahedral(o: vec2<f32>) -> vec3<f32> {
  var v = vec3<f32>(o.x, o.y, 1.0 - abs(o.x) - abs(o.y));
  if (v.z < 0.0) {
    v = vec3<f32>((1.0 - abs(v.yx)) * signNotZero(v.xy), v.z);
  }
  return normalize(v);
}
```

Utility Module

1 file = 1 module

```
fn signNotZero(xy: vec2<f32>) -> vec2<f32> {
  let s = sign(xy);
  return select(s, vec2<f32>(1.0, 1.0), s == vec2<f32>(0.0));
}

/** Assumes that v is a unit vector. The result is an octahedral vector on the [-1, +1] square. */
@export fn encodeOctahedral(v: vec3<f32>) -> vec2<f32> {
  let l1norm = abs(v.x) + abs(v.y) + abs(v.z);
  var result = v.xy * (1.0 / l1norm);
  if (v.z < 0.0) {
    result = (1.0 - abs(result.yx)) * signNotZero(result.xy);
  }
  return result;
}

/** Returns a unit vector. Argument o is an octahedral vector packed via encodeOctahedral,
    on the [-1, +1] square */
@export fn decodeOctahedral(o: vec2<f32>) -> vec3<f32> {
  var v = vec3<f32>(o.x, o.y, 1.0 - abs(o.x) - abs(o.y));
  if (v.z < 0.0) {
    v = vec3<f32>((1.0 - abs(v.yx)) * signNotZero(v.xy), v.z);
  }
  return normalize(v);
}
```

Utility Module

1 file = 1 module

```
fn signNotZero(xy: vec2<f32>) -> vec2<f32> {
  let s = sign(xy);
  return select(s, vec2<f32>(1.0, 1.0), s == vec2<f32>(0.0));
}

/** Assumes that v is a unit vector. The result is an octahedral vector on the [-1, +1] square. */
@export fn encodeOctahedral(v: vec3<f32>) -> vec2<f32> {
  let l1norm = abs(v.x) + abs(v.y) + abs(v.z);
  var result = v.xy * (1.0 / l1norm);
  if (v.z < 0.0) {
    result = (1.0 - abs(result.yx)) * signNotZero(result.xy);
  }
  return result;
}

/** Returns a unit vector. Argument o is an octahedral vector packed via encodeOctahedral,
    on the [-1, +1] square */
@export fn decodeOctahedral(o: vec2<f32>) -> vec3<f32> {
  var v = vec3<f32>(o.x, o.y, 1.0 - abs(o.x) - abs(o.y));
  if (v.z < 0.0) {
    v = vec3<f32>((1.0 - abs(v.yx)) * signNotZero(v.xy), v.z);
  }
  return normalize(v);
}
```

```
use '@use-gpu/wgsl/use/view'::{ getViewPosition, clipToWorld, to3D };
use '@use-gpu/wgsl/use/types'::{ Light, SurfaceFragment };
use '@use-gpu/wgsl/codec/octahedral'::{ decodeOctahedral };



@export fn getDeferredLightFragment(
  uv: vec2<f32>,
  index: u32,
) -> vec4<f32> {
  let albedo = getAlbedo(uv);
  let normal = getNormal(uv);
  let material = getMaterial(uv);
  let depth = getDepth(uv);

  let position = to3D(clipToWorld(vec4<f32>((uv * 2.0 - 1.0) * vec2<f32>(1.0, -1.0), depth, 1.0)));

  let surface = SurfaceFragment(
    vec4<f32>(position, 1.0),
    vec4<f32>(decodeOctahedral(normal.xy), 0.0),
    vec4<f32>(albedo.xyz, 1.0),
    vec4<f32>(0.0),
    material,
```

Rust-like **use syntax**

```
use '@use-gpu/wgsl/use/view'::{ getViewPosition, clipToWorld, to3D };
use '@use-gpu/wgsl/use/types'::{ Light, SurfaceFragment };
use '@use-gpu/wgsl/codec/octahedral'::{ decodeOctahedral };

@export fn getDeferredLightFragment(
  uv: vec2<f32>,
  index: u32,
) -> vec4<f32> {
  let albedo = getAlbedo(uv);
  let normal = getNormal(uv);
  let material = getMaterial(uv);
  let depth = getDepth(uv);

  let position = to3D(clipToWorld(vec4<f32>((uv * 2.0 - 1.0) * vec2<f32>(1.0, -1.0), depth, 1.0)));

  let surface = SurfaceFragment(
    vec4<f32>(position, 1.0),
    vec4<f32>(decodeOctahedral(normal.xy), 0.0),
    vec4<f32>(albedo.xyz, 1.0),
    vec4<f32>(0.0),
    material,
```

```
use '@use-gpu/wgsl/use/view'::{ getViewPosition, clipToWorld, to3D };
use '@use-gpu/wgsl/use/types'::{ Light, SurfaceFragment };
use '@use-gpu/wgsl/codec/octahedral'::{ decodeOctahedral };
```

Rust-like **use syntax**

{ symbol **as** symbol, ... }

```
@export fn getDeferredLightFragment(
  uv: vec2<f32>,
  index: u32,
) -> vec4<f32> {
  let albedo = getAlbedo(uv);
  let normal = getNormal(uv);
  let material = getMaterial(uv);
  let depth = getDepth(uv);

  let position = to3D(clipToWorld(vec4<f32>((uv * 2.0 - 1.0) * vec2<f32>(1.0, -1.0), depth, 1.0)));

  let surface = SurfaceFragment(
    vec4<f32>(position, 1.0),
    vec4<f32>(decodeOctahedral(normal.xy), 0.0),
    vec4<f32>(albedo.xyz, 1.0),
    vec4<f32>(0.0),
    material,
```

```
use '@use-gpu/wgsl/use/view'::{ getViewPosition, clipToWorld, to3D };
use '@use-gpu/wgsl/use/types'::{ Light, SurfaceFragment };
use '@use-gpu/wgsl/codec/octahedral'::{ decodeOctahedral };
```

Rust-like **use syntax**

{ symbol **as** symbol, ... }

Funcs, types, variables

```
@export fn getDeferredLightFragment(
  uv: vec2<f32>,
  index: u32,
) -> vec4<f32> {
  let albedo = getAlbedo(uv);
  let normal = getNormal(uv);
  let material = getMaterial(uv);
  let depth = getDepth(uv);

  let position = to3D(clipToWorld(vec4<f32>((uv * 2.0 - 1.0) * vec2<f32>(1.0, -1.0), depth, 1.0)));

  let surface = SurfaceFragment(
    vec4<f32>(position, 1.0),
    vec4<f32>(decodeOctahedral(normal.xy), 0.0),
    vec4<f32>(albedo.xyz, 1.0),
    vec4<f32>(0.0),
    material,
```

```
use '@use-gpu/wgsl/use/view'::{ getViewPosition, clipToWorld, to3D };
use '@use-gpu/wgsl/use/types'::{ Light, SurfaceFragment };
use '@use-gpu/wgsl/codec/octahedral'::{ decodeOctahedral };
```

Rust-like **use syntax**

{ symbol **as** symbol, ... }

Funcs, types, variables

Recursive exports

```
@export fn getDeferredLightFragment(
  uv: vec2<f32>,
  index: u32,
) -> vec4<f32> {
  let albedo = getAlbedo(uv);
  let normal = getNormal(uv);
  let material = getMaterial(uv);
  let depth = getDepth(uv);

  let position = to3D(clipToWorld(vec4<f32>((uv * 2.0 - 1.0) * vec2<f32>(1.0, -1.0), depth, 1.0)));

  let surface = SurfaceFragment(
    vec4<f32>(position, 1.0),
    vec4<f32>(decodeOctahedral(normal.xy), 0.0),
    vec4<f32>(albedo.xyz, 1.0),
    vec4<f32>(0.0),
    material,
```

Output

```
421
422 fn _0c_clipToWorld(position: vec4<f32>) → vec4<f32> {
423   return _0c_viewUniforms.inverseProjectionViewMatrix * position;
424 }
425
426
427 //// @link fragment/deferred-light _0d_
428
429 fn _0d_getDeferredLightFragment(
430   uv: vec2<f32>,
431   index: u32,
432 ) → vec4<f32> {
433   let albedo = _03_getAlbedo(uv);
434   let normal = _03_getNormal(uv);
435   let material = _03_getMaterial(uv);
436   let depth = _03_getDepth(uv);
437
438   let position = _0c_to3D(_0c_clipToWorld(vec4<f32>((uv * 2.0 - 1.0) * vec2<f32>(1.0, -1.0), depth, 1.0)));
439
440   let surface = _05_SurfaceFragment(
441     vec4<f32>(position, 1.0),
442     vec4<f32>(_07_decodeOctahedral(normal.xy), 0.0),
443     vec4<f32>(albedo.xyz, 1.0),
444     vec4<f32>(0.0),
445     material,
446     albedo.w,
447     0.0,
```

Output

```
421
422 fn _0c_clipToWorld(position: vec4<f32>) → vec4<f32> {
423   return _0c_viewUniforms.inverseProjectionViewMatrix * position;
424 }
425
426
427 //// @link fragment/deferred-light _0d_
428
429 fn _0d_getDeferredLightFragment(
430   uv: vec2<f32>,
431   index: u32,
432 ) → vec4<f32> {
433   let albedo = _03_getAlbedo(uv);
434   let normal = _03_getNormal(uv);
435   let material = _03_getMaterial(uv);
436   let depth = _03_getDepth(uv);
437
438   let position = _0c_to3D(_0c_clipToWorld(vec4<f32>((uv * 2.0 - 1.0) * vec2<f32>(1.0, -1.0), depth, 1.0)));
439
440   let surface = _05_SurfaceFragment(
441     vec4<f32>(position, 1.0),
442     vec4<f32>(_07_decodeOctahedral(normal.xy), 0.0),
443     vec4<f32>(albedo.xyz, 1.0),
444     vec4<f32>(0.0),
445     material,
446     albedo.w,
447     0.0,
```

Modules concatenated with prefix

Output

```
421
422 fn _0c_clipToWorld(position: vec4<f32>) → vec4<f32> {
423   return _0c_viewUniforms.inverseProjectionViewMatrix * position;
424 }
425
426
427 //// @link fragment/deferred-light _0d_
428
429 fn _0d_getDeferredLightFragment(
430   uv: vec2<f32>,
431   index: u32,
432 ) → vec4<f32> {
433   let albedo = _03_getAlbedo(uv);
434   let normal = _03_getNormal(uv);
435   let material = _03_getMaterial(uv);
436   let depth = _03_getDepth(uv);
437
438   let position = _0c_to3D(_0c_clipToWorld(vec4<f32>((uv * 2.0 - 1.0) * vec2<f32>(1.0, -1.0), depth, 1.0)));
439
440   let surface = _05_SurfaceFragment(
441     vec4<f32>(position, 1.0),
442     vec4<f32>(_07_decodeOctahedral(normal.xy), 0.0),
443     vec4<f32>(albedo.xyz, 1.0),
444     vec4<f32>(0.0),
445     material,
446     albedo.w,
447     0.0,
```

Modules concatenated with prefix

All **global symbols** renamed

Output

```
421
422 fn _0c_clipToWorld(position: vec4<f32>) → vec4<f32> {
423   return _0c_viewUniforms.inverseProjectionViewMatrix * position;
424 }
425
426
427 //// @link fragment/deferred-light _0d_
428
429 fn _0d_getDeferredLightFragment(
430   uv: vec2<f32>,
431   index: u32,
432 ) → vec4<f32> {
433   let albedo = _03_getAlbedo(uv);
434   let normal = _03_getNormal(uv);
435   let material = _03_getMaterial(uv);
436   let depth = _03_getDepth(uv);
437
438   let position = _0c_to3D(_0c_clipToWorld(vec4<f32>((uv * 2.0 - 1.0) * vec2<f32>(1.0, -1.0), depth, 1.0)));
439
440   let surface = _05_SurfaceFragment(
441     vec4<f32>(position, 1.0),
442     vec4<f32>(_07_decodeOctahedral(normal.xy), 0.0),
443     vec4<f32>(albedo.xyz, 1.0),
444     vec4<f32>(0.0),
445     material,
446     albedo.w,
447     0.0,
```

Modules concatenated with prefix

All **global symbols** renamed

+ optional minify

Root Shader (Solid Vertex)

```
use '@use-gpu/wgsl/use/types'::{ SolidVertex };

@link fn getVertex(v: u32, i: u32) -> SolidVertex {};
@optional @link fn toColorSpace(c: vec4<f32>) -> vec4<f32> { return c; }

struct VertexOutput {
  @builtin(position) position: vec4<f32>,
  @location(0) fragColor: vec4<f32>,
  @location(1) fragUV: vec4<f32>,
  @location(2) fragST: vec4<f32>,
  @location(3) fragScissor: vec4<f32>,
};

@vertex
fn main(
  @builtin(vertex_index) vertexIndex: u32,
  @builtin(instance_index) instanceIndex: u32,
) -> VertexOutput {
  let v = getVertex(vertexIndex, instanceIndex);

  return VertexOutput(
    v.position,
    toColorSpace(v.color),
    v.uv,
    v.st,
    v.scissor,
  );
}
```

Root Shader (Solid Vertex)

```wgsl
use '@use-gpu/wgsl/use/types'::{ SolidVertex };

@link fn getVertex(v: u32, i: u32) -> SolidVertex {};
@optional @link fn toColorSpace(c: vec4<f32>) -> vec4<f32> { return c; }

struct VertexOutput {
  @builtin(position) position: vec4<f32>,
  @location(0) fragColor: vec4<f32>,
  @location(1) fragUV: vec4<f32>,
  @location(2) fragST: vec4<f32>,
  @location(3) fragScissor: vec4<f32>,
};

@vertex
fn main(
  @builtin(vertex_index) vertexIndex: u32,
  @builtin(instance_index) instanceIndex: u32,
) -> VertexOutput {
  let v = getVertex(vertexIndex, instanceIndex);

  return VertexOutput(
    v.position,
    toColorSpace(v.color),
    v.uv,
    v.st,
    v.scissor,
  );
}
```

Thin wrapper around a **getVertex(...)**

Root Shader (Solid Vertex)

```
1  use '@use-gpu/wgsl/use/types'::{ SolidVertex };
2
3  @link fn getVertex(v: u32, i: u32) -> SolidVertex {};
4  @optional @link fn toColorSpace(c: vec4<f32>) -> vec4<f32> { return c; }
5
6  struct VertexOutput {
7    @builtin(position) position: vec4<f32>,
8    @location(0) fragColor: vec4<f32>,
9    @location(1) fragUV: vec4<f32>,
10   @location(2) fragST: vec4<f32>,
11   @location(3) fragScissor: vec4<f32>,
12 };
13
14 @vertex
15 fn main(
16   @builtin(vertex_index) vertexIndex: u32,
17   @builtin(instance_index) instanceIndex: u32,
18 ) -> VertexOutput {
19   let v = getVertex(vertexIndex, instanceIndex);
20
21   return VertexOutput(
22     v.position,
23     toColorSpace(v.color),
24     v.uv,
25     v.st,
26     v.scissor,
27   );
28 }
```

Thin wrapper around a **getVertex(...)**

Optional color space conversion

Root Shader (Shaded Vertex)

```
use '@use-gpu/wgsl/use/types'::{ ShadedVertex };

@link fn getVertex(v: u32, i: u32) -> ShadedVertex {};
@optional @link fn toColorSpace(c: vec4<f32>) -> vec4<f32> { return c; }

struct VertexOutput {
  @builtin(position) position: vec4<f32>,
  @location(0) fragColor: vec4<f32>,
  @location(1) fragUV: vec4<f32>,
  @location(2) fragST: vec4<f32>,
  @location(3) fragNormal: vec4<f32>,
  @location(4) fragTangent: vec4<f32>,
  @location(5) fragPosition: vec4<f32>,
  @location(6) fragScissor: vec4<f32>,
};

@vertex
fn main(
  @builtin(vertex_index) vertexIndex: u32,
  @builtin(instance_index) instanceIndex: u32,
) -> VertexOutput {
  let v = getVertex(vertexIndex, instanceIndex);

  return VertexOutput(
    v.position,
    toColorSpace(v.color),
    v.uv,
    v.st,
    v.normal,
```

Root Shader (Shaded Vertex)

```
use '@use-gpu/wgsl/use/types'::{ ShadedVertex };

@link fn getVertex(v: u32, i: u32) -> ShadedVertex {};
@optional @link fn toColorSpace(c: vec4<f32>) -> vec4<f32> { return c; }

struct VertexOutput {
  @builtin(position) position: vec4<f32>,
  @location(0) fragColor: vec4<f32>,
  @location(1) fragUV: vec4<f32>,
  @location(2) fragST: vec4<f32>,
  @location(3) fragNormal: vec4<f32>,
  @location(4) fragTangent: vec4<f32>,
  @location(5) fragPosition: vec4<f32>,
  @location(6) fragScissor: vec4<f32>,
};

@vertex
fn main(
  @builtin(vertex_index) vertexIndex: u32,
  @builtin(instance_index) instanceIndex: u32,
) -> VertexOutput {
  let v = getVertex(vertexIndex, instanceIndex);

  return VertexOutput(
    v.position,
    toColorSpace(v.color),
    v.uv,
    v.st,
    v.normal,
```

Thin wrapper around a **getVertex(...)**

Root Shader (Shaded Vertex)

```wgsl
use '@use-gpu/wgsl/use/types'::{ ShadedVertex };

@link fn getVertex(v: u32, i: u32) -> ShadedVertex {};
@optional @link fn toColorSpace(c: vec4<f32>) -> vec4<f32> { return c; }

struct VertexOutput {
  @builtin(position) position: vec4<f32>,
  @location(0) fragColor: vec4<f32>,
  @location(1) fragUV: vec4<f32>,
  @location(2) fragST: vec4<f32>,
  @location(3) fragNormal: vec4<f32>,
  @location(4) fragTangent: vec4<f32>,
  @location(5) fragPosition: vec4<f32>,
  @location(6) fragScissor: vec4<f32>,
};

@vertex
fn main(
  @builtin(vertex_index) vertexIndex: u32,
  @builtin(instance_index) instanceIndex: u32,
) -> VertexOutput {
  let v = getVertex(vertexIndex, instanceIndex);

  return VertexOutput(
    v.position,
    toColorSpace(v.color),
    v.uv,
    v.st,
    v.normal,
```

Thin wrapper around a **getVertex(…)**

More **attributes**

Root Shader (Picking Vertex)

```
use '@use-gpu/wgsl/use/types'::{ PickVertex };

@link fn getVertex(v: u32, i: u32) -> PickVertex {};
@optional @link fn getPicking(i: u32) -> vec2<u32> { return vec2<u32>(0u, 0u); };

struct VertexOutput {
  @builtin(position) position: vec4<f32>,
  @location(0) fragScissor: vec4<f32>,
  @location(1) fragUV: vec2<f32>,
  @location(2) @interpolate(flat) fragId: u32,
  @location(3) @interpolate(flat) fragIndex: u32,
};

@vertex
fn main(
  @builtin(vertex_index) vertexIndex: u32,
  @builtin(instance_index) instanceIndex: u32,
) -> VertexOutput {
  var v = getVertex(vertexIndex, instanceIndex);
  var p = getPicking(v.index);

  return VertexOutput(
    v.position,
    v.scissor,
    v.uv.xy,
    p.x,
    p.y,
  );
}
```

Root Shader (Picking Vertex)

Thin wrapper around
a **getVertex(...)**

```
use '@use-gpu/wgsl/use/types'::{ PickVertex };

@link fn getVertex(v: u32, i: u32) -> PickVertex {};
@optional @link fn getPicking(i: u32) -> vec2<u32> { return vec2<u32>(0u, 0u); };

struct VertexOutput {
  @builtin(position) position: vec4<f32>,
  @location(0) fragScissor: vec4<f32>,
  @location(1) fragUV: vec2<f32>,
  @location(2) @interpolate(flat) fragId: u32,
  @location(3) @interpolate(flat) fragIndex: u32,
};

@vertex
fn main(
  @builtin(vertex_index) vertexIndex: u32,
  @builtin(instance_index) instanceIndex: u32,
) -> VertexOutput {
  var v = getVertex(vertexIndex, instanceIndex);
  var p = getPicking(v.index);

  return VertexOutput(
    v.position,
    v.scissor,
    v.uv.xy,
    p.x,
    p.y,
  );
}
```

Root Shader (Picking Vertex)

```wgsl
use '@use-gpu/wgsl/use/types'::{ PickVertex };

@link fn getVertex(v: u32, i: u32) -> PickVertex {};
@optional @link fn getPicking(i: u32) -> vec2<u32> { return vec2<u32>(0u, 0

struct VertexOutput {
  @builtin(position) position: vec4<f32>,
  @location(0) fragScissor: vec4<f32>,
  @location(1) fragUV: vec2<f32>,
  @location(2) @interpolate(flat) fragId: u32,
  @location(3) @interpolate(flat) fragIndex: u32,
};

@vertex
fn main(
  @builtin(vertex_index) vertexIndex: u32,
  @builtin(instance_index) instanceIndex: u32,
) -> VertexOutput {
  var v = getVertex(vertexIndex, instanceIndex);
  var p = getPicking(v.index);

  return VertexOutput(
    v.position,
    v.scissor,
    v.uv.xy,
    p.x,
    p.y,
  );
}
```

Thin wrapper around a **getVertex(...)**

Picking info via **getPicking(v.index)**

Root Shader (Picking Vertex)

```
use '@use-gpu/wgsl/use/types'::{ PickVertex };

@link fn getVertex(v: u32, i: u32) -> PickVertex {};
@optional @link fn getPicking(i: u32) -> vec2<u32> { return vec2<u32>(0u, 0

struct VertexOutput {
  @builtin(position) position: vec4<f32>,
  @location(0) fragScissor: vec4<f32>,
  @location(1) fragUV: vec2<f32>,
  @location(2) @interpolate(flat) fragId: u32,
  @location(3) @interpolate(flat) fragIndex: u32,
};

@vertex
fn main(
  @builtin(vertex_index) vertexIndex: u32,
  @builtin(instance_index) instanceIndex: u32,
) -> VertexOutput {
  var v = getVertex(vertexIndex, instanceIndex);
  var p = getPicking(v.index);

  return VertexOutput(
    v.position,
    v.scissor,
    v.uv.xy,
    p.x,
    p.y,
  );
}
```

Thin wrapper around a **getVertex(...)**

Picking info via **getPicking(v.index)**

Different **attributes**

Root Shader (Picking Vertex)

```
use '@use-gpu/wgsl/use/types'::{ PickVertex };

@link fn getVertex(v: u32, i: u32) -> PickVertex {};
@optional @link fn getPicking(i: u32) -> vec2<u32> { return vec2<u32>(0u, 0

struct VertexOutput {
  @builtin(position) position: vec4<f32>,
  @location(0) fragScissor: vec4<f32>,
  @location(1) fragUV: vec2<f32>,
  @location(2) @interpolate(flat) fragId: u32,
  @location(3) @interpolate(flat) fragIndex: u32,
};

@vertex
fn main(
  @builtin(vertex_index) vertexIndex: u32,
  @builtin(instance_index) instanceIndex: u32,
) -> VertexOutput {
  var v = getVertex(vertexIndex, instanceIndex);
  var p = getPicking(v.index);

  return VertexOutput(
    v.position,
    v.scissor,
    v.uv.xy,
    p.x,
    p.y,
  );
}
```

Thin wrapper around a **getVertex(...)**

Picking info via **getPicking(v.index)**

Different **attributes**

Needs **matching `@fragment`**

Root Shader (Picking Vertex)

```wgsl
use '@use-gpu/wgsl/use/types'::{ PickVertex };

@link fn getVertex(v: u32, i: u32) -> PickVertex {};
@optional @link fn getPicking(i: u32) -> vec2<u32> { return vec2<u32>(0u, 0

struct VertexOutput {
  @builtin(position) position: vec4<f32>,
  @location(0) fragScissor: vec4<f32>,
  @location(1) fragUV: vec2<f32>,
  @location(2) @interpolate(flat) fragId: u32,
  @location(3) @interpolate(flat) fragIndex: u32,
};

@vertex
fn main(
  @builtin(vertex_index) vertexIndex: u32,
  @builtin(instance_index) instanceIndex: u32,
) -> VertexOutput {
  var v = getVertex(vertexIndex, instanceIndex);
  var p = getPicking(v.index);

  return VertexOutput(
    v.position,
    v.scissor,
    v.uv.xy,
    p.x,
    p.y,
  );
}
```

Thin wrapper around a **getVertex(...)**

Picking info via **getPicking(v.index)**

Different **attributes**

Needs **matching `@fragment`**

Permutations 😔

Root Shader (Picking Vertex)

```wgsl
use '@use-gpu/wgsl/use/types'::{ PickVertex };

@link fn getVertex(v: u32, i: u32) -> PickVertex {};
@optional @link fn getPicking(i: u32) -> vec2<u32> { return vec2<u32>(0u, 0

struct VertexOutput {
  @builtin(position) position: vec4<f32>,
  @location(0) fragScissor: vec4<f32>,
  @location(1) fragUV: vec2<f32>,
  @location(2) @interpolate(flat) fragId: u32,
  @location(3) @interpolate(flat) fragIndex: u32,
};

@vertex
fn main(
  @builtin(vertex_index) vertexIndex: u32,
  @builtin(instance_index) instanceIndex: u32,
) -> VertexOutput {
  var v = getVertex(vertexIndex, instanceIndex);
  var p = getPicking(v.index);

  return VertexOutput(
    v.position,
    v.scissor,
    v.uv.xy,
    p.x,
    p.y,
  );
}
```

Thin wrapper around a **getVertex(...)**

Picking info via **getPicking(v.index)**

Different **attributes**

Needs **matching `@fragment`**

Permutations 😔

eg. **UI Shader** for SDFs

Root Shader (Picking Vertex)

```wgsl
use '@use-gpu/wgsl/use/types'::{ PickVertex };

@link fn getVertex(v: u32, i: u32) -> PickVertex {};
@optional @link fn getPicking(i: u32) -> vec2<u32> { return vec2<u32>(0u, 0

struct VertexOutput {
  @builtin(position) position: vec4<f32>,
  @location(0) fragScissor: vec4<f32>,
  @location(1) fragUV: vec2<f32>,
  @location(2) @interpolate(flat) fragId: u32,
  @location(3) @interpolate(flat) fragIndex: u32,
};

@vertex
fn main(
  @builtin(vertex_index) vertexIndex: u32,
  @builtin(instance_index) instanceIndex: u32,
) -> VertexOutput {
  var v = getVertex(vertexIndex, instanceIndex);
  var p = getPicking(v.index);

  return VertexOutput(
    v.position,
    v.scissor,
    v.uv.xy,
    p.x,
    p.y,
  );
}
```

Thin wrapper around a **getVertex(...)**

Picking info via **getPicking(v.index)**

Different **attributes**

Needs **matching `@fragment`**

Permutations 😔

eg. **UI Shader** for SDFs

Hints at **Entity-Component** style?

Dynamic Linking - Direct (Read/Write)

```
use '@use-gpu/wgsl/use/array'::{ sizeToModulus2, packIndex2, wrapIndex2 };

@link fn getSize() -> vec2<u32> {};

@link var<storage> curlBuffer: array<f32>;

@link var<storage, read_write> velocityBufferOut: array<vec4<f32>>;

@link var<storage> velocityBufferInPnHat: array<vec4<f32>>;
@link var<storage> velocityBufferInPn1Hat: array<vec4<f32>>;
@link var<storage> velocityBufferInPn: array<vec4<f32>>;

@compute @workgroup_size(8, 8)
fn main(
  @builtin(global_invocation_id) globalId: vec3<u32>,
) {
  let size = getSize();
  if (any(globalId.xy >= size)) { return; }
  let fragmentId = globalId.xy;

  let modulus = sizeToModulus2(size);
  let center = packIndex2(fragmentId, modulus);

  let pn = velocityBufferInPn[center];
  let pn1hat = velocityBufferInPn1Hat[center];
```

Dynamic Linking - Direct (Read/Write)

```
use '@use-gpu/wgsl/use/array'::{ sizeToModulus2, packIndex2, wrapIndex2 };

@link fn getSize() -> vec2<u32> {};

@link var<storage> curlBuffer: array<f32>;

@link var<storage, read_write> velocityBufferOut: array<vec4<f32>>;

@link var<storage> velocityBufferInPnHat: array<vec4<f32>>;
@link var<storage> velocityBufferInPn1Hat: array<vec4<f32>>;
@link var<storage> velocityBufferInPn: array<vec4<f32>>;

@compute @workgroup_size(8, 8)
fn main(
  @builtin(global_invocation_id) globalId: vec3<u32>,
) {
  let size = getSize();
  if (any(globalId.xy >= size)) { return; }
  let fragmentId = globalId.xy;

  let modulus = sizeToModulus2(size);
  let center = packIndex2(fragmentId, modulus);

  let pn = velocityBufferInPn[center];
  let pn1hat = velocityBufferInPn1Hat[center];
```

Linked **directly to binding**

Dynamic Linking - Indirect

```
use '@use-gpu/wgsl/use/view'::{ getViewPosition, clipToWorld, to3D };
use '@use-gpu/wgsl/use/types'::{ Light, SurfaceFragment };
use '@use-gpu/wgsl/codec/octahedral'::{ decodeOctahedral };

@link fn getAlbedo(uv: vec2<f32>) -> vec4<f32>;
@link fn getNormal(uv: vec2<f32>) -> vec4<f32>;
@link fn getMaterial(uv: vec2<f32>) -> vec4<f32>;
@link fn getEmissive(uv: vec2<f32>) -> vec4<f32>;
@link fn getDepth(uv: vec2<f32>) -> f32;

@link fn getLight(i: u32) -> Light;

@link fn applyLight(
  N: vec3<f32>,
  V: vec3<f32>,
  light: Light,
  surface: SurfaceFragment,
) -> vec3<f32>;

@export fn getDeferredLightFragment(
  uv: vec2<f32>,
  index: u32,
```

Dynamic Linking - Indirect

```
use '@use-gpu/wgsl/use/view'::{ getViewPosition, clipToWorld, to3D };
use '@use-gpu/wgsl/use/types'::{ Light, SurfaceFragment };
use '@use-gpu/wgsl/codec/octahedral'::{ decodeOctahedral };

@link fn getAlbedo(uv: vec2<f32>) -> vec4<f32>;
@link fn getNormal(uv: vec2<f32>) -> vec4<f32>;
@link fn getMaterial(uv: vec2<f32>) -> vec4<f32>;
@link fn getEmissive(uv: vec2<f32>) -> vec4<f32>;
@link fn getDepth(uv: vec2<f32>) -> f32;

@link fn getLight(i: u32) -> Light;

@link fn applyLight(
  N: vec3<f32>,
  V: vec3<f32>,
  light: Light,
  surface: SurfaceFragment,
) -> vec3<f32>;

@export fn getDeferredLightFragment(
  uv: vec2<f32>,
  index: u32,
```

Function to function

Dynamic Linking - Indirect

```
use '@use-gpu/wgsl/use/view'::{ getViewPosition, clipToWorld, to3D };
use '@use-gpu/wgsl/use/types'::{ Light, SurfaceFragment };
use '@use-gpu/wgsl/codec/octahedral'::{ decodeOctahedral };
```

```
@link fn getAlbedo(uv: vec2<f32>) -> vec4<f32>;
@link fn getNormal(uv: vec2<f32>) -> vec4<f32>;
@link fn getMaterial(uv: vec2<f32>) -> vec4<f32>;
@link fn getEmissive(uv: vec2<f32>) -> vec4<f32>;
@link fn getDepth(uv: vec2<f32>) -> f32;

@link fn getLight(i: u32) -> Light;
```

Polymorphic accessors

```
@link fn applyLight(
  N: vec3<f32>,
  V: vec3<f32>,
  light: Light,
  surface: SurfaceFragment,
) -> vec3<f32>;
```

Function to function

```
@export fn getDeferredLightFragment(
  uv: vec2<f32>,
  index: u32,
```

Dynamic Linking - Indirect

```
use '@use-gpu/wgsl/use/view'::{ getViewPosition, clipToWorld, to3D };
use '@use-gpu/wgsl/use/types'::{ Light, SurfaceFragment };
use '@use-gpu/wgsl/codec/octahedral'::{ decodeOctahedral };
```

```
@link fn getAlbedo(uv: vec2<f32>) -> vec4<f32>;
@link fn getNormal(uv: vec2<f32>) -> vec4<f32>;
@link fn getMaterial(uv: vec2<f32>) -> vec4<f32>;
@link fn getEmissive(uv: vec2<f32>) -> vec4<f32>;
@link fn getDepth(uv: vec2<f32>) -> f32;

@link fn getLight(i: u32) -> Light;
```

```
@link fn applyLight(
  N: vec3<f32>,
  V: vec3<f32>,
  light: Light,
  surface: SurfaceFragment,
) -> vec3<f32>;
```

```
@export fn getDeferredLightFragment(
  uv: vec2<f32>,
  index: u32,
```

Polymorphic accessors

- Constant
- Storage array
- Texture
- Closure

Function to function

```
//// @link @access [getAlbedo getNormal getMaterial getEmissive getDepth] [7m9eatfvw8] _03_

@group(2) @binding(3) var _03_getAlbedoTexture: texture_2d<f32>;
@group(2) @binding(4) var _03_getAlbedoSampler: sampler;

fn _03_getAlbedo(a: vec2<f32>) -> vec4<f32> {
  return textureSample(_03_getAlbedoTexture, _03_getAlbedoSampler, a);
}

@group(2) @binding(5) var _03_getNormalTexture: texture_2d<f32>;
@group(2) @binding(6) var _03_getNormalSampler: sampler;

fn _03_getNormal(a: vec2<f32>) -> vec4<f32> {
  return textureSample(_03_getNormalTexture, _03_getNormalSampler, a);
}

@group(2) @binding(7) var _03_getMaterialTexture: texture_2d<f32>;
@group(2) @binding(8) var _03_getMaterialSampler: sampler;

fn _03_getMaterial(a: vec2<f32>) -> vec4<f32> {
  return textureSample(_03_getMaterialTexture, _03_getMaterialSampler, a);
}

@group(2) @binding(9) var _03_getEmissiveTexture: texture_2d<f32>;
@group(2) @binding(10) var _03_getEmissiveSampler: sampler;

fn _03_getEmissive(a: vec2<f32>) -> vec4<f32> {
  return textureSample(_03_getEmissiveTexture, _03_getEmissiveSampler, a);
}

@group(2) @binding(11) var _03_getDepthTexture: texture_depth_2d;
@group(2) @binding(12) var _03_getDepthSampler: sampler;
```

```
91
92  //// @link @access [getAlbedo getNormal getMaterial getEmissive getDepth] [7m9eatfvw8] _03_
93
94  @group(2) @binding(3) var _03_getAlbedoTexture: texture_2d<f32>;
95  @group(2) @binding(4) var _03_getAlbedoSampler: sampler;
96
97  fn _03_getAlbedo(a: vec2<f32>) → vec4<f32> {
98    return textureSample(_03_getAlbedoTexture, _03_getAlbedoSampler, a);
99  }
100
101 @group(2) @binding(5) var _03_getNormalTexture: texture_2d<f32>;
102 @group(2) @binding(6) var _03_getNormalSampler: sampler;
103
104 fn _03_getNormal(a: vec2<f32>) → vec4<f32> {
105   return textureSample(_03_getNormalTexture, _03_getNormalSampler, a);
106 }
107
108 @group(2) @binding(7) var _03_getMaterialTexture: texture_2d<f32>;
109 @group(2) @binding(8) var _03_getMaterialSampler: sampler;
110
111 fn _03_getMaterial(a: vec2<f32>) → vec4<f32> {
112   return textureSample(_03_getMaterialTexture, _03_getMaterialSampler, a);
113 }
114
115 @group(2) @binding(9) var _03_getEmissiveTexture: texture_2d<f32>;
116 @group(2) @binding(10) var _03_getEmissiveSampler: sampler;
117
118 fn _03_getEmissive(a: vec2<f32>) → vec4<f32> {
119   return textureSample(_03_getEmissiveTexture, _03_getEmissiveSampler, a);
120 }
121
122 @group(2) @binding(11) var _03_getDepthTexture: texture_depth_2d;
123 @group(2) @binding(12) var _03_getDepthSampler: sampler;
```

Auto-generated bindings

```
91
92  //// @link @access [getAlbedo getNormal getMaterial getEmissive getDepth] [7m9eatfvw8] _03_
93
94  @group(2) @binding(3) var _03_getAlbedoTexture: texture_2d<f32>;
95  @group(2) @binding(4) var _03_getAlbedoSampler: sampler;
96
97  fn _03_getAlbedo(a: vec2<f32>) → vec4<f32> {
98    return textureSample(_03_getAlbedoTexture, _03_getAlbedoSampler, a);
99  }
100
101 @group(2) @binding(5) var _03_getNormalTexture: texture_2d<f32>;
102 @group(2) @binding(6) var _03_getNormalSampler: sampler;
103
104 fn _03_getNormal(a: vec2<f32>) → vec4<f32> {
105   return textureSample(_03_getNormalTexture, _03_getNormalSampler, a);
106 }
107
108 @group(2) @binding(7) var _03_getMaterialTexture: texture_2d<f32>;
109 @group(2) @binding(8) var _03_getMaterialSampler: sampler;
110
111 fn _03_getMaterial(a: vec2<f32>) → vec4<f32> {
112   return textureSample(_03_getMaterialTexture, _03_getMaterialSampler, a);
113 }
114
115 @group(2) @binding(9) var _03_getEmissiveTexture: texture_2d<f32>;
116 @group(2) @binding(10) var _03_getEmissiveSampler: sampler;
117
118 fn _03_getEmissive(a: vec2<f32>) → vec4<f32> {
119   return textureSample(_03_getEmissiveTexture, _03_getEmissiveSampler, a);
120 }
121
122 @group(2) @binding(11) var _03_getDepthTexture: texture_depth_2d;
123 @group(2) @binding(12) var _03_getDepthSampler: sampler;
```

Auto-generated bindings

Auto-generated accessors

```
//// @link @virtual _01_

struct _VT_Type {
  _VT_1_getInstanceSize: u32,
  _VT_2_getAlbedo: vec4<f32>,
  _VT_2_getEmissive: vec3<f32>,
  _VT_2_getMetalness: f32,
  _VT_2_getRoughness: f32,
};
@group(1) @binding(5) var<uniform> _VT_Uniform: _VT_Type;

//// @link @access [getAlbedo getEmissive getMetalness getRoughness] [hw29vdl6ww] _VT_2_

fn _VT_2_getAlbedo() -> vec4<f32> {
  return _VT_Uniform._VT_2_getAlbedo;
}

fn _VT_2_getEmissive() -> vec3<f32> {
  return _VT_Uniform._VT_2_getEmissive;
}

fn _VT_2_getMetalness() -> f32 {
  return _VT_Uniform._VT_2_getMetalness;
}

fn _VT_2_getRoughness() -> f32 {
  return _VT_Uniform._VT_2_getRoughness;
}
```

```
19
20  //// @link @virtual _01_
21
22  struct _VT_Type {
23    _VT_1_getInstanceSize: u32,
24    _VT_2_getAlbedo: vec4<f32>,
25    _VT_2_getEmissive: vec3<f32>,
26    _VT_2_getMetalness: f32,
27    _VT_2_getRoughness: f32,
28  };
29  @group(1) @binding(5) var<uniform> _VT_Uniform: _VT_Type;
30
31  //// @link @access [getAlbedo getEmissive getMetalness getRoughness] [hw29vdl6ww] _VT_2_
32
33  fn _VT_2_getAlbedo() → vec4<f32> {
34    return _VT_Uniform._VT_2_getAlbedo;
35  }
36
37  fn _VT_2_getEmissive() → vec3<f32> {
38    return _VT_Uniform._VT_2_getEmissive;
39  }
40
41  fn _VT_2_getMetalness() → f32 {
42    return _VT_Uniform._VT_2_getMetalness;
43  }
44
45  fn _VT_2_getRoughness() → f32 {
46    return _VT_Uniform._VT_2_getRoughness;
47  }
48
```

Auto-generated uniforms

```
19
20  //// @link @virtual _01_
21
22  struct _VT_Type {
23    _VT_1_getInstanceSize: u32,
24    _VT_2_getAlbedo: vec4<f32>,
25    _VT_2_getEmissive: vec3<f32>,
26    _VT_2_getMetalness: f32,
27    _VT_2_getRoughness: f32,
28  };
29  @group(1) @binding(5) var<uniform> _VT_Uniform: _VT_Type;
30
31  //// @link @access [getAlbedo getEmissive getMetalness getRoughness] [hw29vdl6ww] _VT_2_
32
33  fn _VT_2_getAlbedo() → vec4<f32> {
34    return _VT_Uniform._VT_2_getAlbedo;
35  }
36
37  fn _VT_2_getEmissive() → vec3<f32> {
38    return _VT_Uniform._VT_2_getEmissive;
39  }
40
41  fn _VT_2_getMetalness() → f32 {
42    return _VT_Uniform._VT_2_getMetalness;
43  }
44
45  fn _VT_2_getRoughness() → f32 {
46    return _VT_Uniform._VT_2_getRoughness;
47  }
48
```

Auto-generated uniforms

Auto-generated accessors

```
//// @link @virtual _01_

struct _VT_Type {
  _VT_1_getInstanceSize: u32,
  _VT_2_getAlbedo: vec4<f32>,
  _VT_2_getEmissive: vec3<f32>,
  _VT_2_getMetalness: f32,
  _VT_2_getRoughness: f32,
};
@group(1) @binding(5) var<uniform> _VT_Uniform: _VT_Type;

//// @link @access [getAlbedo getEmissive getMetalness getRoughness] [hw29vdl6ww] _VT_2_

fn _VT_2_getAlbedo() -> vec4<f32> {
  return _VT_Uniform._VT_2_getAlbedo;
}

fn _VT_2_getEmissive() -> vec3<f32> {
  return _VT_Uniform._VT_2_getEmissive;
}

fn _VT_2_getMetalness() -> f32 {
  return _VT_Uniform._VT_2_getMetalness;
}

fn _VT_2_getRoughness() -> f32 {
  return _VT_Uniform._VT_2_getRoughness;
}
```

Auto-generated uniforms

Auto-generated accessors

Two bind groups:
static vs volatile

```
//// @link @virtual _01_

struct _VT_Type {
  _VT_1_getInstanceSize: u32,
  _VT_2_getAlbedo: vec4<f32>,
  _VT_2_getEmissive: vec3<f32>,
  _VT_2_getMetalness: f32,
  _VT_2_getRoughness: f32,
};
@group(1) @binding(5) var<uniform> _VT_Uniform: _VT_Type;

//// @link @access [getAlbedo getEmissive getMetalness getRoughness] [hw29vdl6ww] _VT_2_

fn _VT_2_getAlbedo() -> vec4<f32> {
  return _VT_Uniform._VT_2_getAlbedo;
}

fn _VT_2_getEmissive() -> vec3<f32> {
  return _VT_Uniform._VT_2_getEmissive;
}

fn _VT_2_getMetalness() -> f32 {
  return _VT_Uniform._VT_2_getMetalness;
}

fn _VT_2_getRoughness() -> f32 {
  return _VT_Uniform._VT_2_getRoughness;
}
```

Auto-generated uniforms

Auto-generated accessors

Two bind groups:
static vs volatile

Other 2 for custom use

Type Inference (1 way)

```
use '@use-gpu/wgsl/use/types'::{ Light };
use '@use-gpu/wgsl/fragment/pbr'::{ PBR };

@infer type T;
@link fn applyLight(
  N: vec3<f32>,
  V: vec3<f32>,
  light: Light,
  @infer(T) surface: T,
) -> vec3<f32> {}

@link fn getLightCount() -> u32;
@link fn getLight(index: u32) -> Light;

@export fn applyLights(
  N: vec3<f32>,
  V: vec3<f32>,
  surface: T,
) -> vec3<f32> {

  var radiance: vec3<f32> = vec3<f32>(0.0);

  let lightCount = getLightCount();
  let n = min(lightCount, 1024u);
  for (var i = 0u; i < lightCount; i++) {
    let light = getLight(i);
    let r = applyLight(N, V, light, surface);
```

Type Inference (1 way)

```
use '@use-gpu/wgsl/use/types'::{ Light };
use '@use-gpu/wgsl/fragment/pbr'::{ PBR };

@infer type T;
@link fn applyLight(
  N: vec3<f32>,
  V: vec3<f32>,
  light: Light,
  @infer(T) surface: T,
) -> vec3<f32> {}

@link fn getLightCount() -> u32;
@link fn getLight(index: u32) -> Light;

@export fn applyLights(
  N: vec3<f32>,
  V: vec3<f32>,
  surface: T,
) -> vec3<f32> {

  var radiance: vec3<f32> = vec3<f32>(0.0);

  let lightCount = getLightCount();
  let n = min(lightCount, 1024u);
  for (var i = 0u; i < lightCount; i++) {
    let light = getLight(i);
    let r = applyLight(N, V, light, surface);
```

Infer **type T** from
linked function argument

Type Inference (1 way)

```
use '@use-gpu/wgsl/use/types'::{ Light };
use '@use-gpu/wgsl/fragment/pbr'::{ PBR };

@infer type T;
@link fn applyLight(
  N: vec3<f32>,
  V: vec3<f32>,
  light: Light,
  @infer(T) surface: T,
) -> vec3<f32> {}

@link fn getLightCount() -> u32;
@link fn getLight(index: u32) -> Light;

@export fn applyLights(
  N: vec3<f32>,
  V: vec3<f32>,
  surface: T,
) -> vec3<f32> {

  var radiance: vec3<f32> = vec3<f32>(0.0);

  let lightCount = getLightCount();
  let n = min(lightCount, 1024u);
  for (var i = 0u; i < lightCount; i++) {
    let light = getLight(i);
    let r = applyLight(N, V, light, surface);
```

Infer **type T** from **linked function** argument

Or **return type T**

Type Inference (1 way)

```
use '@use-gpu/wgsl/use/types'::{ Light };
use '@use-gpu/wgsl/fragment/pbr'::{ PBR };

@infer type T;
@link fn applyLight(
  N: vec3<f32>,
  V: vec3<f32>,
  light: Light,
  @infer(T) surface: T,
) -> vec3<f32> {}

@link fn getLightCount() -> u32;
@link fn getLight(index: u32) -> Light;

@export fn applyLights(
  N: vec3<f32>,
  V: vec3<f32>,
  surface: T,
) -> vec3<f32> {

  var radiance: vec3<f32> = vec3<f32>(0.0);

  let lightCount = getLightCount();
  let n = min(lightCount, 1024u);
  for (var i = 0u; i < lightCount; i++) {
    let light = getLight(i);
    let r = applyLight(N, V, light, surface);
```

Infer **type T** from **linked function** argument

Or **return type T**

Function is **generic** over **surface type**

Type Inference (1 way)

```
use '@use-gpu/wgsl/use/types'::{ Light };
use '@use-gpu/wgsl/fragment/pbr'::{ PBR };

@infer type T;
@link fn applyLight(
  N: vec3<f32>,
  V: vec3<f32>,
  light: Light,
  @infer(T) surface: T,
) -> vec3<f32> {}

@link fn getLightCount() -> u32;
@link fn getLight(index: u32) -> Light;

@export fn applyLights(
  N: vec3<f32>,
  V: vec3<f32>,
  surface: T,
) -> vec3<f32> {

  var radiance: vec3<f32> = vec3<f32>(0.0);

  let lightCount = getLightCount();
  let n = min(lightCount, 1024u);
  for (var i = 0u; i < lightCount; i++) {
    let light = getLight(i);
    let r = applyLight(N, V, light, surface);
```

Infer **type T** from **linked function** argument

Or **return type T**

Function is **generic** over **surface type**

Duck typing = valid **if it compiles** ✅

Type Inference (1 way)

```
use '@use-gpu/wgsl/use/types'::{ Light, SurfaceFragment };

@link fn applyMaterial(
  N: vec3<f32>,
  L: vec3<f32>,
  V: vec3<f32>,
  surface: SurfaceFragment,
) -> vec3<f32> {}

@optional @link fn applyDirectionalShadow(
  light: Light,
  surface: SurfaceFragment,
) -> f32 { return 1.0; }

@optional @link fn applyPointShadow(
  light: Light,
  surface: SurfaceFragment,
) -> f32 { return 1.0; }

@export fn applyLight(
  N: vec3<f32>,
  V: vec3<f32>,
  light: Light,
  surface: SurfaceFragment,
) -> vec3<f32> {
  var L: vec3<f32>;

  var intensity: f32 = light.intensity * 3.1415;
```

Type Inference (1 way)

```
1  use '@use-gpu/wgsl/use/types'::{ Light, SurfaceFragment };
2
3  @link fn applyMaterial(
4    N: vec3<f32>,
5    L: vec3<f32>,
6    V: vec3<f32>,
7    surface: SurfaceFragment,
8  ) -> vec3<f32> {}
9
10 @optional @link fn applyDirectionalShadow(
11   light: Light,
12   surface: SurfaceFragment,
13 ) -> f32 { return 1.0; }
14
15 @optional @link fn applyPointShadow(
16   light: Light,
17   surface: SurfaceFragment,
18 ) -> f32 { return 1.0; }
19
20 @export fn applyLight(
21   N: vec3<f32>,
22   V: vec3<f32>,
23   light: Light,
24   surface: SurfaceFragment,
25 ) -> vec3<f32> {
26   var L: vec3<f32>;
27
28   var intensity: f32 = light.intensity * 3.1415;
```

Type inferred **from here**

Type Inference (1 way)

```
1  use '@use-gpu/wgsl/use/types'::{ Light, SurfaceFragment };
2
3  @link fn applyMaterial(
4    N: vec3<f32>,
5    L: vec3<f32>,
6    V: vec3<f32>,
7    surface: SurfaceFragment,
8  ) -> vec3<f32> {}
9
10 @optional @link fn applyDirectionalShadow(
11   light: Light,
12   surface: SurfaceFragment,
13 ) -> f32 { return 1.0; }
14
15 @optional @link fn applyPointShadow(
16   light: Light,
17   surface: SurfaceFragment,
18 ) -> f32 { return 1.0; }
19
20 @export fn applyLight(
21   N: vec3<f32>,
22   V: vec3<f32>,
23   light: Light,
24   surface: SurfaceFragment,
25 ) -> vec3<f32> {
26   var L: vec3<f32>;
27
28   var intensity: f32 = light.intensity * 3.1415;
```

Lighting independent of material

Type inferred **from here**

Type Inference (1 way)

```
1  use '@use-gpu/wgsl/use/types'::{ Light, SurfaceFragment };
2
3  @link fn applyMaterial(
4    N: vec3<f32>,
5    L: vec3<f32>,
6    V: vec3<f32>,
7    surface: SurfaceFragment,
8  ) -> vec3<f32> {}
9
10 @optional @link fn applyDirectionalShadow(
11   light: Light,
12   surface: SurfaceFragment,
13 ) -> f32 { return 1.0; }
14
15 @optional @link fn applyPointShadow(
16   light: Light,
17   surface: SurfaceFragment,
18 ) -> f32 { return 1.0; }
19
20 @export fn applyLight(
21   N: vec3<f32>,
22   V: vec3<f32>,
23   light: Light,
24   surface: SurfaceFragment,
25 ) -> vec3<f32> {
26   var L: vec3<f32>;
27
28   var intensity: f32 = light.intensity * 3.1415;
```

Lighting independent of material

Shadow sampler is injected and optional

Type inferred **from here**

Wireframe Decorator

```
6
7   @link fn getVertex(v: u32, i: u32) -> SolidVertex {};
8   @link fn getInstanceSize() -> u32 {};
9
10  @export fn getWireframeListVertex(vertexIndex: u32, instanceIndex: u32) -> SolidVertex {
11    var ij = getStripIndex(vertexIndex);
12    var xy = vec2<f32>(ij) * 2.0 - 1.0;
13
14    var n = getInstanceSize();
15    var f = instanceIndex % n;
16    var i = instanceIndex / n;
17
18    var v = u32(f) % 3u;
19    var t = u32(f) - v;
20
21    var ia: u32;
22    var ib: u32;
23    var ic: u32;
24    if (v == 0u) {
25      ia = t;
26      ib = t + 1...
```

Wireframe Decorator

```
6
7   @link fn getVertex(v: u32, i: u32) -> SolidVertex {};
8   @link fn getInstanceSize() -> u32 {};
9
10  @export fn getWireframeListVertex(vertexIndex: u32, instanceIndex: u32) -> SolidVertex {
11    var ij = getStripIndex(vertexIndex);
12    var xy = vec2<f32>(ij) * 2.0 - 1.0;
13
14    var n = getInstanceSize();
15    var f = instanceIndex % n;
16    var i = instanceIndex / n;
17
18    var v = u32(f) % 3u;
19    var t = u32(f) - v;
20
21    var ia: u32;
22    var ib: u32;
23    var ic: u32;
24    if (v == 0u) {
25      ia = t;
```

Wrapper around
a **getVertex(...)**

Wireframe Decorator

```
6
7   @link fn getVertex(v: u32, i: u32) -> SolidVertex {};
8   @link fn getInstanceSize() -> u32 {};
9
10  @export fn getWireframeListVertex(vertexIndex: u32, instanceIndex: u32) -> SolidVertex {
11    var ij = getStripIndex(vertexIndex);
12    var xy = vec2<f32>(ij) * 2.0 - 1.0;
13
14    var n = getInstanceSize();
15    var f = instanceIndex % n;
16    var i = instanceIndex / n;
17
18    var v = u32(f) % 3u;
19    var t = u32(f) - v;
20
21    var ia: u32;
22    var ib: u32;
23    var ic: u32;
24    if (v == 0u) {
25      ia = t;
```

Wrapper around
a **getVertex(…)**

Is a **getVertex(…)**

Wireframe Decorator

```
6
7   @link fn getVertex(v: u32, i: u32) -> SolidVertex {};
8   @link fn getInstanceSize() -> u32 {};
9
10  @export fn getWireframeListVertex(vertexIndex: u32, instanceIndex: u32) -> SolidVertex {
11    var ij = getStripIndex(vertexIndex);
12    var xy = vec2<f32>(ij) * 2.0 - 1.0;
13
14    var n = getInstanceSize();
15    var f = instanceIndex % n;
16    var i = instanceIndex / n;
17
18    var v = u32(f) % 3u;
19    var t = u32(f) - v;
20
21    var ia: u32;
22    var ib: u32;
23    var ic: u32;
24    if (v == 0u) {
25      ia = t;
```

Wrapper around
a **getVertex(...)**

Is a **getVertex(...)**

Adjusted `vertexCount` / `instanceCount` on CPU

Wireframe Decorator

```
6
7   @link fn getVertex(v: u32, i: u32) -> SolidVertex {};
8   @link fn getInstanceSize() -> u32 {};
9
10  @export fn getWireframeListVertex(vertexIndex: u32, instanceIndex: u32) -> SolidVertex {
11    var ij = getStripIndex(vertexIndex);
12    var xy = vec2<f32>(ij) * 2.0 - 1.0;
13
14    var n = getInstanceSize();
15    var f = instanceIndex % n;
16    var i = instanceIndex / n;
17
18    var v = u32(f) % 3u;
19    var t = u32(f) - v;
20
21    var ia: u32;
22    var ib: u32;
23    var ic: u32;
24    if (v == 0u) {
25      ia = t;
```

Operate on **indices**

Wrapper around
a **getVertex(...)**

Is a **getVertex(...)**

Adjusted `vertexCount` / `instanceCount` on CPU

Wireframe Decorator

```
6
7   @link fn getVertex(v: u32, i: u32) -> SolidVertex {};
8   @link fn getInstanceSize() -> u32 {};
9
10  @export fn getWireframeListVertex(vertexIndex: u32, instanceIndex: u32) -> SolidVertex {
11    var ij = getStripIndex(vertexIndex);
12    var xy = vec2<f32>(ij) * 2.0 - 1.0;
13
14    var n = getInstanceSize();
15    var f = instanceIndex % n;
16    var i = instanceIndex / n;
17
18    var v = u32(f) % 3u;
19    var t = u32(f) - v;
20
21    var ia: u32;
22    var ib: u32;
23    var ic: u32;
24    if (v == 0u) {
25      ia = t;
```

Operate on **indices**

Wrapper around
a **getVertex(...)**

Is a **getVertex(...)**

Adjusted `vertexCount` / `instanceCount` on CPU

```
39
40    var a = getVertex(ia, i);
41    var b = getVertex(ib, i);
42    var c = getVertex(ic, i);
43
```

Call 'vertex shader' **3 times** in vertex shader

Wireframe Decorator

```
@link fn getVertex(v: u32, i: u32) -> SolidVertex {};
@link fn getInstanceSize() -> u32 {};

@export fn getWireframeListVertex(vertexIndex: u32, instanceIndex: u32) -> SolidVertex {
  var ij = getStripIndex(vertexIndex);
  var xy = vec2<f32>(ij) * 2.0 - 1.0;

  var n = getInstanceSize();
  var f = instanceIndex % n;
  var i = instanceIndex / n;

  var v = u32(f) % 3u;
  var t = u32(f) - v;

  var ia: u32;
  var ib: u32;
  var ic: u32;
  if (v == 0u) {
    ia = t;
```

Operate on **indices**

```
    var a = getVertex(ia, i);
    var b = getVertex(ib, i);
    var c = getVertex(ic, i);
```

Call 'vertex shader' **3 times** in vertex shader

Wrapper around
a **getVertex(...)**

Is a **getVertex(...)**

Adjusted `vertexCount` / `instanceCount` on CPU

```
  var left  = a.position.xyz /
  var right = b.position.xyz / b.position.w;
  var other = c.position.xyz / c.position.w;

  let lineWidth = getViewPixelRatio() * 2.0;
  var join: vec3<f32>;
  if (ij.x > 0u) {
    join = getLineJoin(left, right, other, f32(ij.x) - 1.0, xy.y, lineWidth, 3, 0);
  }
  else {
    join = getLineJoin(other, left, right, 1.0, xy.y, lineWidth, 3, 0);
  }

  return SolidVertex(
    vec4<f32>(join, 1.0),
    vec4<f32>(1.0),
    vec4<f32>(0.0),
                .0),
                .0),

}
```

Wireframe Decorator

```
@link fn getVertex(v: u32, i: u32) -> SolidVertex {};
@link fn getInstanceSize() -> u32 {};

@export fn getWireframeListVertex(vertexIndex: u32, instanceIndex: u32) -> SolidVertex {
  var ij = getStripIndex(vertexIndex);
  var xy = vec2<f32>(ij) * 2.0 - 1.0;

  var n = getInstanceSize();
  var f = instanceIndex % n;
  var i = instanceIndex / n;

  var v = u32(f) % 3u;
  var t = u32(f) - v;

  var ia: u32;
  var ib: u32;
  var ic: u32;
  if (v == 0u) {
    ia = t;
```

Operate on **indices**

Wrapper around a **getVertex(...)**

Is a **getVertex(...)**

Adjusted `vertexCount` / `instanceCount` on CPU

```
    var a = getVertex(ia, i);
    var b = getVertex(ib, i);
    var c = getVertex(ic, i);
```

Call 'vertex shader' **3 times** in vertex shader

```
  var left  = a.position.xyz /
  var right = b.position.xyz / b.position.w;
  var other = c.position.xyz / c.position.w;

  let lineWidth = getViewPixelRatio() * 2.0;
  var join: vec3<f32>;
  if (ij.x > 0u) {
    join = getLineJoin(left, right, other, f32(ij.x) - 1.0, xy.y, lineWidth, 3, 0);
  }
  else {
    join = getLineJoin(other, left, right, 1.0, xy.y, lineWidth, 3, 0);
  }

  return SolidVertex(
    vec4<f32>(join, 1.0),
    vec4<f32>(1.0),
    vec4<f32>(0.0),
                .0),
                .0),

}
```

Re-use **line join** logic

Wireframe Decorator

```
@link fn getVertex(v: u32, i: u32) -> SolidVertex {};
@link fn getInstanceSize() -> u32 {};

@export fn getWireframeListVertex(vertexIndex: u32, instanceIndex: u32) -> SolidVertex {
  var ij = getStripIndex(vertexIndex);
  var xy = vec2<f32>(ij) * 2.0 - 1.0;

  var n = getInstanceSize();
  var f = instanceIndex % n;
  var i = instanceIndex / n;

  var v = u32(f) % 3u;
  var t = u32(f) - v;

  var ia: u32;
  var ib: u32;
  var ic: u32;
  if (v == 0u) {
    ia = t;
```

Operate on **indices**

```
    var a = getVertex(ia, i);
    var b = getVertex(ib, i);
    var c = getVertex(ic, i);
```

Call 'vertex shader' **3 times** in vertex shader

Wrapper around
a **getVertex(...)**

Is a **getVertex(...)**

Adjusted `vertexCount` / `instanceCount` on CPU

```
  var left  = a.position.xyz /
  var right = b.position.xyz / b.position.w;
  var other = c.position.xyz / c.position.w;

  let lineWidth = getViewPixelRatio() * 2.0;
  var join: vec3<f32>;
  if (ij.x > 0u) {
    join = getLineJoin(left, right, other, f32(ij.x) - 1.0, xy.y, lineWidth, 3, 0);
  }
  else {
    join = getLineJoin(other, left, right, 1.0, xy.y, lineWidth, 3, 0);
  }

  return SolidVertex(
    vec4<f32>(join, 1.0),
    vec4<f32>(1.0),
    vec4<f32>(0.0),
    .0),
    .0),
}
```

Re-use **line join** logic

Any **sub-type of**
`SolidVertex` works 🤷
(by accident... type is not mentioned)

Compute Shader Wireframe

```
@link var<storage, read>       sourceCommand: array<u32>;
@link var<storage, read_write> destinationCommand: array<u32>;

@compute @workgroup_size(1)
fn main() {
  let vertexCount = sourceCommand[0];
  let instanceCount = sourceCommand[1];

  if (isTriangleStrip) {
    let edges = (vertexCount - 2) * 2 + 1;
    destinationCommand[0] = 4;
    destinationCommand[1] = edges * instanceCount;
    destinationCommand[64] = edges;
  }
  else {
    destinationCommand[0] = 18;
    destinationCommand[1] = vertexCount * instanceCount;
    destinationCommand[64] = vertexCount;
  }
}
```

Compute Shader Wireframe

```
@link var<storage, read>       sourceCommand: array<u32>;
@link var<storage, read_write> destinationCommand: array<u32>;

@compute @workgroup_size(1)
fn main() {
  let vertexCount = sourceCommand[0];
  let instanceCount = sourceCommand[1];

  if (isTriangleStrip) {
    let edges = (vertexCount - 2) * 2 + 1;
    destinationCommand[0] = 4;
    destinationCommand[1] = edges * instanceCount;
    destinationCommand[64] = edges;
  }
  else {
    destinationCommand[0] = 18;
    destinationCommand[1] = vertexCount * instanceCount;
    destinationCommand[64] = vertexCount;
  }
}
```

Convert **triangle** draw
to **line** draw

Compute Shader Wireframe

```
@link var<storage, read>       sourceCommand: array<u32>;
@link var<storage, read_write> destinationCommand: array<u32>;

@compute @workgroup_size(1)
fn main() {
  let vertexCount = sourceCommand[0];
  let instanceCount = sourceCommand[1];

  if (isTriangleStrip) {
    let edges = (vertexCount - 2) * 2 + 1;
    destinationCommand[0] = 4;
    destinationCommand[1] = edges * instanceCount;
    destinationCommand[64] = edges;
  }
  else {
    destinationCommand[0] = 18;
    destinationCommand[1] = vertexCount * instanceCount;
    destinationCommand[64] = vertexCount;
  }
}
```

Convert **triangle** draw
to **line** draw

Compute new
vertex & instance count

Compute Shader Wireframe

```
@link var<storage, read>       sourceCommand: array<u32>;
@link var<storage, read_write> destinationCommand: array<u32>;

@compute @workgroup_size(1)
fn main() {
  let vertexCount = sourceCommand[0];
  let instanceCount = sourceCommand[1];

  if (isTriangleStrip) {
    let edges = (vertexCount - 2) * 2 + 1;
    destinationCommand[0] = 4;
    destinationCommand[1] = edges * instanceCount;
    destinationCommand[64] = edges;
  }
  else {
    destinationCommand[0] = 18;
    destinationCommand[1] = vertexCount * instanceCount;
    destinationCommand[64] = vertexCount;
  }
}
```

Convert **triangle** draw to **line** draw

Compute new **vertex & instance count**

Set **`instanceSize`**

Compute Shader Wireframe

```
@link var<storage, read>       sourceCommand: array<u32>;
@link var<storage, read_write> destinationCommand: array<u32>;

@compute @workgroup_size(1)
fn main() {
  let vertexCount = sourceCommand[0];
  let instanceCount = sourceCommand[1];

  if (isTriangleStrip) {
    let edges = (vertexCount - 2) * 2 + 1;
    destinationCommand[0] = 4;
    destinationCommand[1] = edges * instanceCount;
    destinationCommand[64] = edges;
  }
  else {
    destinationCommand[0] = 18;
    destinationCommand[1] = vertexCount * instanceCount;
    destinationCommand[64] = vertexCount;
  }
}

//// @link @access [getInstanceSize] [b6bkz0vmdk] _0d_

@group(2) @binding(0) var<storage> _0d_getInstanceSizeStorage: array<u32>;

fn _0d_getInstanceSize() -> u32 {
  return _0d_getInstanceSizeStorage[0u];
}
```

Convert **triangle** draw to **line** draw

Compute new **vertex & instance count**

Set **`instanceSize`**

Compute Shader Wireframe

```
@link var<storage, read>       sourceCommand: array<u32>;
@link var<storage, read_write> destinationCommand: array<u32>;

@compute @workgroup_size(1)
fn main() {
  let vertexCount = sourceCommand[0];
  let instanceCount = sourceCommand[1];

  if (isTriangleStrip) {
    let edges = (vertexCount - 2) * 2 + 1;
    destinationCommand[0] = 4;
    destinationCommand[1] = edges * instanceCount;
    destinationCommand[64] = edges;
  }
  else {
    destinationCommand[0] = 18;
    destinationCommand[1] = vertexCount * instanceCount;
    destinationCommand[64] = vertexCount;
  }
}

//// @link @access [getInstanceSize] [b6bkz0vmdk] _0d_

@group(2) @binding(0) var<storage> _0d_getInstanceSizeStorage: array<u32>;

fn _0d_getInstanceSize() -> u32 {
  return _0d_getInstanceSizeStorage[0u];
}
```

Convert **triangle** draw to **line** draw

Compute new **vertex & instance count**

Set **`instanceSize`**

Storage bound @ [64]

Compute Shader Wireframe

```
@link var<storage, read>       sourceCommand: array<u32>;
@link var<storage, read_write> destinationCommand: array<u32>;

@compute @workgroup_size(1)
fn main() {
  let vertexCount = sourceCommand[0];
  let instanceCount = sourceCommand[1];

  if (isTriangleStrip) {
    let edges = (vertexCount - 2) * 2 + 1;
    destinationCommand[0] = 4;
    destinationCommand[1] = edges * instanceCount;
    destinationCommand[64] = edges;
  }
  else {
    destinationCommand[0] = 18;
    destinationCommand[1] = vertexCount * instanceCount;
    destinationCommand[64] = vertexCount;
  }
}
```

```
//// @link @access [getInstanceSize] [b6bkz0vmdk] _0d_

@group(2) @binding(0) var<storage> _0d_getInstanceSizeStorage: array<u32>;

fn _0d_getInstanceSize() -> u32 {
  return _0d_getInstanceSizeStorage[0u];
}
```

Convert **triangle** draw to **line** draw

Compute new **vertex & instance count**

Set **`instanceSize`**

Storage bound @ [64]

`getInstanceSize` lambda

Constants = 'Feature Flags'

```
11  const HAS_INDICES = true;
12  const HAS_SEGMENTS = false;
13  const FLAT_NORMALS = false;
14  const UNWELDED_COLORS = false;
15  const UNWELDED_NORMALS = false;
16  const UNWELDED_TANGENTS = false;
17  const UNWELDED_UVS = false;
18  const UNWELDED_LOOKUPS = false;
19
```

```
305    var normal: vec4<f32>;
306    if (FLAT_NORMALS) {
307      let b = _0b_positions(beforeIndex).xyz;
308      let a = _0b_positions(afterIndex).xyz;
309      normal = vec4<f32>(normalize(cross(vertex.xyz - a, vertex.xyz - b)), 1.0);
310    }
311    else {
312      normal = _0b_normals(normalIndex);
313    }
314
```

Constants = 'Feature Flags'

```
11  const HAS_INDICES = true;
12  const HAS_SEGMENTS = false;
13  const FLAT_NORMALS = false;
14  const UNWELDED_COLORS = false;
15  const UNWELDED_NORMALS = false;
16  const UNWELDED_TANGENTS = false;
17  const UNWELDED_UVS = false;
18  const UNWELDED_LOOKUPS = false;
19
```

GLSL **`#define`** equivalent

= injected as global **`const`** when binding any module

Hoisted to top

```
305   var normal: vec4<f32>;
306   if (FLAT_NORMALS) {
307     let b = _0b_positions(beforeIndex).xyz;
308     let a = _0b_positions(afterIndex).xyz;
309     normal = vec4<f32>(normalize(cross(vertex.xyz - a, vertex.xyz - b)), 1.0);
310   }
311   else {
312     normal = _0b_normals(normalIndex);
313   }
314
```

Constants = 'Feature Flags'

```
11  const HAS_INDICES = true;
12  const HAS_SEGMENTS = false;
13  const FLAT_NORMALS = false;
14  const UNWELDED_COLORS = false;
15  const UNWELDED_NORMALS = false;
16  const UNWELDED_TANGENTS = false;
17  const UNWELDED_UVS = false;
18  const UNWELDED_LOOKUPS = false;
19
```

GLSL **`#define`** equivalent
= injected as global **`const`**
when binding any module

Hoisted to top

```
305    var normal: vec4<f32>;
306    if (FLAT_NORMALS) {
307      let b = _0b_positions(beforeIndex).xyz;
308      let a = _0b_positions(afterIndex).xyz;
309      normal = vec4<f32>(normalize(cross(vertex.xyz - a, vertex.xyz - b)), 1.0);
310    }
311    else {
312      normal = _0b_normals(normalIndex);
313    }
314
```

Const condition,
should be optimized away?

Constants = 'Feature Flags'

```
11  const HAS_INDICES = true;
12  const HAS_SEGMENTS = false;
13  const FLAT_NORMALS = false;
14  const UNWELDED_COLORS = false;
15  const UNWELDED_NORMALS = false;
16  const UNWELDED_TANGENTS = false;
17  const UNWELDED_UVS = false;
18  const UNWELDED_LOOKUPS = false;
19
```

GLSL **`#define`** equivalent
= injected as global **`const`**
when binding any module

Hoisted to top

```
305    var normal: vec4<f32>;
306    if (FLAT_NORMALS) {
307      let b = _0b_positions(beforeIndex).xyz;
308      let a = _0b_positions(afterIndex).xyz;
309      normal = vec4<f32>(normalize(cross(vertex.xyz - a, vertex.xyz - b)), 1.0);
310    }
311    else {
312      normal = _0b_normals(normalIndex);
313    }
314
```

Const condition,
should be optimized away?

Not scoped, but **too convenient** not to have 🤷

Constants = 'Feature Flags'

```
11  const HAS_INDICES = true;
12  const HAS_SEGMENTS = false;
13  const FLAT_NORMALS = false;
14  const UNWELDED_COLORS = false;
15  const UNWELDED_NORMALS = false;
16  const UNWELDED_TANGENTS = false;
17  const UNWELDED_UVS = false;
18  const UNWELDED_LOOKUPS = false;
19
```

```
305    var normal: vec4<f32>;
306    if (FLAT_NORMALS) {
307      let b = _0b_positions(beforeIndex).xyz;
308      let a = _0b_positions(afterIndex).xyz;
309      normal = vec4<f32>(normalize(cross(vertex.xyz - a, vertex.xyz
310    }
311    else {
312      normal = _0b_normals(normalIndex);
313    }
314
```

Not scoped, but **too convenient** not to have 🤷

GLSL **`#define`** equivalent
= injected as global **`const`**
when binding any module

Hoisted to top

Const condition,
should be optimized away?

Nobody fully 'owns'
pipeline flags, e.g.

- **culling** mode (CW vs CCW)
- **dither** or **premultiplied** alpha
- per-pixel / SDF **scissor**

Constants = 'Feature Flags'

```
11  const HAS_INDICES = true;
12  const HAS_SEGMENTS = false;
13  const FLAT_NORMALS = false;
14  const UNWELDED_COLORS = false;
15  const UNWELDED_NORMALS = false;
16  const UNWELDED_TANGENTS = false;
17  const UNWELDED_UVS = false;
18  const UNWELDED_LOOKUPS = false;
19
```

```
305    var normal: vec4<f32>;
306    if (FLAT_NORMALS) {
307      let b = _0b_positions(beforeIndex).xyz;
308      let a = _0b_positions(afterIndex).xyz;
309      normal = vec4<f32>(normalize(cross(vertex.xyz - a, vertex.xyz
310    }
311    else {
312      normal = _0b_
313    }
314
```

GLSL **`#define`** equivalent
= injected as global **`const`**
when binding any module

Hoisted to top

Const condition,
should be optimized away?

Spoiler: *WebGPU cubemaps are backwards*

Nobody fully 'owns' pipeline flags, e.g.

- **culling** mode (CW vs CCW)
- **dither** or **premultiplied** alpha
- per-pixel / SDF **scissor**

Not scoped, but **too convenient** not to have 🤷

Point-free Style

`op(bundle, bundle)` in TS

Point-free Style

op(bundle, bundle) in TS

```
96  //// @link @cast [positions vec3<f32>] _0d_
97
98  fn _0d_cast(a: u32) → vec4<f32> {
99    let v = _0a_positions(a);
100   return vec4<f32>(v.xyz, 1.0);
101 }
```

```
130 fn _0h_cast(a: u32) → vec4<f32> {
131   let v = _0f_sizes(a);
132   return vec4<f32>(-v, -v, v, v) * 0.5;
133 }
```

Point-free Style

`op(bundle, bundle)` in TS

```
96  //// @link @cast [positions vec3<f32>] _0d_
97
98  fn _0d_cast(a: u32) → vec4<f32> {
99    let v = _0a_positions(a);
100   return vec4<f32>(v.xyz, 1.0);
101 }

130 fn _0h_cast(a: u32) → vec4<f32> {
131   let v = _0f_sizes(a);
132   return vec4<f32>(-v, -v, v, v) * 0.5;
133 }
```

Can do **vec# auto-casts** and **extended swizzles.**

Point-free Style

`op(bundle, bundle)` in TS

```
96  //// @link @cast [positions vec3<f32>] _0d_
97
98  fn _0d_cast(a: u32) → vec4<f32> {
99    let v = _0a_positions(a);
100   return vec4<f32>(v.xyz, 1.0);
101 }

130 fn _0h_cast(a: u32) → vec4<f32> {
131   let v = _0f_sizes(a);
132   return vec4<f32>(-v, -v, v, v) * 0.5;
133 }

122 //// @link @chain [chain] _0g_
123
124 fn _0g_chain(a: u32) → vec4<f32> {
125   return _0e_getCartesianPosition(_0d_cast(a));
126 }

75  fn _07_chain(a: vec4<f32>, b: vec4<f32>, c: vec4<f32>) → vec4<f32> {
76    return _06_getPassThruColor(_05_getMaskedColor(a, b, c), b, c);
77  }
```

Can do **vec# auto-casts** and **extended swizzles.**

Point-free Style

`op(bundle, bundle)` in TS

```
96  //// @link @cast [positions vec3<f32>] _0d_
97
98  fn _0d_cast(a: u32) → vec4<f32> {
99    let v = _0a_positions(a);
100   return vec4<f32>(v.xyz, 1.0);
101 }

130 fn _0h_cast(a: u32) → vec4<f32> {
131   let v = _0f_sizes(a);
132   return vec4<f32>(-v, -v, v, v) * 0.5;
133 }

122 //// @link @chain [chain] _0g_
123
124 fn _0g_chain(a: u32) → vec4<f32> {
125   return _0e_getCartesianPosition(_0d_cast(a));
126 }

75  fn _07_chain(a: vec4<f32>, b: vec4<f32>, c: vec4<f32>) → vec4<f32> {
76    return _06_getPassThruColor(_05_getMaskedColor(a, b, c), b, c);
77  }
```

Can do **vec# auto-casts** and **extended swizzles.**

Function **chaining** with **`... rest`.**

Attributes: Quads 

```
use '@use-gpu/wgsl/use/types'::{ SolidVertex };
use '@use-gpu/wgsl/use/view'::{ getViewResolution, worldToClip, getPerspectiveScale, getViewScale, applyZBias };
use '@use-gpu/wgsl/geometry/quad'::{ getQuadUV };

@optional @link fn getPosition(i: u32) -> vec4<f32> { return vec4<f32>(0.0, 0.0, 0.0, 1.0); };
@optional @link fn getScissor(i: u32) -> vec4<f32> { return vec4<f32>(1.0); };

@optional @link fn getRectangle(i: u32) -> vec4<f32> { return vec4<f32>(-1.0, -1.0, 1.0, 1.0); };
@optional @link fn getColor(i: u32) -> vec4<f32> { return vec4<f32>(0.5, 0.5, 0.5, 1.0); };
@optional @link fn getDepth(i: u32) -> f32 { return 0.0; };
@optional @link fn getZBias(i: u32) -> f32 { return 0.0; };
@optional @link fn getUV(i: u32) -> vec4<f32> { return vec4<f32>(0.0, 0.0, 1.0, 1.0); };
@optional @link fn getST(i: u32) -> vec4<f32> { return vec4<f32>(0.5, 0.5, 0.0, 0.0); };

@optional @link fn getPointCount() -> f32 { return 1.0; }

@export fn getQuadVertex(vertexIndex: u32, elementIndex: u32) -> SolidVertex {

  var position = getPosition(elementIndex);
  var scissor = getScissor(elementIndex);
  var rectangle = getRectangle(elementIndex);
  var color = getColor(elementIndex);
  var depth = getDepth(elementIndex);
  var rectangleUV = getUV(elementIndex);
  var st4 = getST(elementIndex);
```

Attributes: Quads

1 file = 1 interface

```
use '@use-gpu/wgsl/use/types'::{ SolidVertex };
use '@use-gpu/wgsl/use/view'::{ getViewResolution, worldToClip, getPerspectiveScale, getViewScale, applyZBias };
use '@use-gpu/wgsl/geometry/quad'::{ getQuadUV };

@optional @link fn getPosition(i: u32) -> vec4<f32> { return vec4<f32>(0.0, 0.0, 0.0, 1.0); };
@optional @link fn getScissor(i: u32) -> vec4<f32> { return vec4<f32>(1.0); };

@optional @link fn getRectangle(i: u32) -> vec4<f32> { return vec4<f32>(-1.0, -1.0, 1.0, 1.0); };
@optional @link fn getColor(i: u32) -> vec4<f32> { return vec4<f32>(0.5, 0.5, 0.5, 1.0); };
@optional @link fn getDepth(i: u32) -> f32 { return 0.0; };
@optional @link fn getZBias(i: u32) -> f32 { return 0.0; };
@optional @link fn getUV(i: u32) -> vec4<f32> { return vec4<f32>(0.0, 0.0, 1.0, 1.0); };
@optional @link fn getST(i: u32) -> vec4<f32> { return vec4<f32>(0.5, 0.5, 0.0, 0.0); };

@optional @link fn getPointCount() -> f32 { return 1.0; }

@export fn getQuadVertex(vertexIndex: u32, elementIndex: u32) -> SolidVertex {

  var position = getPosition(elementIndex);
  var scissor = getScissor(elementIndex);
  var rectangle = getRectangle(elementIndex);
  var color = getColor(elementIndex);
  var depth = getDepth(elementIndex);
  var rectangleUV = getUV(elementIndex);
  var st4 = getST(elementIndex);
  var zBias = getZBias(elementIndex);
```

Attributes: Quads

1 file = 1 interface

```
use '@use-gpu/wgsl/use/types'::{ SolidVertex };
use '@use-gpu/wgsl/use/view'::{ getViewResolution, worldToClip, getPerspectiveScale, getViewScale, applyZBias };
use '@use-gpu/wgsl/geometry/quad'::{ getQuadUV };

@optional @link fn getPosition(i: u32) -> vec4<f32> { return vec4<f32>(0.0, 0.0, 0.0, 1.0); };
@optional @link fn getScissor(i: u32) -> vec4<f32> { return vec4<f32>(1.0); };

@optional @link fn getRectangle(i: u32) -> vec4<f32> { return vec4<f32>(-1.0, -1.0, 1.0, 1.0); };
@optional @link fn getColor(i: u32) -> vec4<f32> { return vec4<f32>(0.5, 0.5, 0.5, 1.0); };
@optional @link fn getDepth(i: u32) -> f32 { return 0.0; };
@optional @link fn getZBias(i: u32) -> f32 { return 0.0; };
@optional @link fn getUV(i: u32) -> vec4<f32> { return vec4<f32>(0.0, 0.0, 1.0, 1.0); };
@optional @link fn getST(i: u32) -> vec4<f32> { return vec4<f32>(0.5, 0.5, 0.0, 0.0); };

@optional @link fn getPointCount() -> f32 { return 1.0; }

@export fn getQuadVertex(vertexIndex: u32, elementIndex: u32) -> SolidVertex {

  var position = getPosition(elementIndex);
  var scissor = getScissor(elementIndex);
  var rectangle = getRectangle(elementIndex);
  var color = getColor(elementIndex);
  var depth = getDepth(elementIndex);
  var rectangleUV = getUV(elementIndex);
  var st4 = getST(elementIndex);
```

Is a **getVertex(...)**

Attributes: Quads

1 file = 1 interface

Any attribute can be uniform or per-element

```
use '@use-gpu/wgsl/use/types'::{ SolidVertex };
use '@use-gpu/wgsl/use/view'::{ getViewResolution, worlo                    applyZBias };
use '@use-gpu/wgsl/geometry/quad'::{ getQuadUV };

@optional @link fn getPosition(i: u32) -> vec4<f32> { return vec4<f32>(0.0, 0.0, 0.0, 1.0); };
@optional @link fn getScissor(i: u32) -> vec4<f32> { return vec4<f32>(1.0); };

@optional @link fn getRectangle(i: u32) -> vec4<f32> { return vec4<f32>(-1.0, -1.0, 1.0, 1.0); };
@optional @link fn getColor(i: u32) -> vec4<f32> { return vec4<f32>(0.5, 0.5, 0.5, 1.0); };
@optional @link fn getDepth(i: u32) -> f32 { return 0.0; };
@optional @link fn getZBias(i: u32) -> f32 { return 0.0; };
@optional @link fn getUV(i: u32) -> vec4<f32> { return vec4<f32>(0.0, 0.0, 1.0, 1.0); };
@optional @link fn getST(i: u32) -> vec4<f32> { return vec4<f32>(0.5, 0.5, 0.0, 0.0); };

@optional @link fn getPointCount() -> f32 { return 1.0; }

@export fn getQuadVertex(vertexIndex: u32, elementIndex: u32) -> SolidVertex {

  var position = getPosition(elementIndex);
  var scissor = getScissor(elementIndex);
  var rectangle = getRectangle(elementIndex);
  var color = getColor(elementIndex);
  var depth = getDepth(elementIndex);
  var rectangleUV = getUV(elementIndex);
  var st4 = getST(elementIndex);
```

Is a **getVertex(...)**

Attributes: Quads

1 file = 1 interface

Any attribute can be uniform or per-element

```
use '@use-gpu/wgsl/use/types'::{ SolidVertex };
use '@use-gpu/wgsl/use/view'::{ getViewResolution, worlo                    applyZBias };
use '@use-gpu/wgsl/geometry/quad'::{ getQuadUV };

@optional @link fn getPosition(i: u32) -> vec4<f32> { return vec4<f32>(0.0, 0.0, 0.0, 1.0); };
@optional @link fn getScissor(i: u32) -> vec4<f32> { return vec4<f32>(1.0); };

@optional @link fn getRectangle(i: u32) -> vec4<f32> { return vec4<f32>(-1.0, -1.0, 1.0, 1.0); };
@optional @link fn getColor(i: u32) -> vec4<f32> { return vec4<f32>(0.5, 0.5, 0.5, 1.0); };
@optional @link fn getDepth(i: u32) -> f32 { return 0.0; };
@optional @link fn getZBias(i: u32) -> f32 { return 0.0; };
@optional @link fn getUV(i: u32) -> vec4<f32> { return vec4<f32>(0.0, 0.0, 1.0, 1.0); };
@optional @link fn getST(i: u32) -> vec4<f32> { return vec4<f32>(0.5, 0.5, 0.0, 0.0); };

@optional @link fn getPointCount() -> f32 { return 1.0;

@export fn getQuadVertex(vertexIndex: u32, elementIndex: u32) -> SolidVertex {

  var position = getPosition(elementIndex);
  var scissor = getScissor(elementIndex);
  var rectangle = getRectangle(elementIndex);
  var color = getColor(elementIndex);
  var depth = getDepth(elementIndex);
  var rectangleUV = getUV(elementIndex);
  var st4 = getST(elementIndex);
```

Default value = fn body

Is a **getVertex(...)**

Attributes: Quads

1 file = 1 interface

Any attribute can be uniform or per-element

```
use '@use-gpu/wgsl/use/types'::{ SolidVertex };
use '@use-gpu/wgsl/use/view'::{ getViewResolution, worlo                    applyZBias };
use '@use-gpu/wgsl/geometry/quad'::{ getQuadUV };

@optional @link fn getPosition(i: u32) -> vec4<f32> { return vec4<f32>(0.0, 0.0, 0.0, 1.0); };
@optional @link fn getScissor(i: u32) -> vec4<f32> { return vec4<f3

@optional @link fn getRectangle(i: u32) -> vec4<f32> { return vec4<
@optional @link fn getColor(i: u32) -> vec4<f32> { return vec4<f32>
@optional @link fn getDepth(i: u32) -> f32 { return 0.0; };
@optional @link fn getZBias(i: u32) -> f32 { return 0.0; };
@optional @link fn getUV(i: u32) -> vec4<f32> { return vec4<f32>(0.
@optional @link fn getST(i: u32) -> vec4<f32> { return vec4<f32>(0.5, 0.5, 0.0, 0.0); };

@optional @link fn getPointCount() -> f32 { return 1.0;

@export fn getQuadVertex(vertexIndex: u32, elementIndex: u32) -> SolidVertex {

  var position = getPosition(elementIndex);
  var scissor = getScissor(elementIndex);
  var rectangle = getRectangle(elementIndex);
  var color = getColor(elementIndex);
  var depth = getDepth(elementIndex);
  var rectangleUV = getUV(elementIndex);
  var st4 = getST(elementIndex);
```

Could potentially be replaced with **1 struct** but what about **defaults** and **spread**?

Default value = fn body

Is a **getVertex(...)**

Attributes: Quads

1 file = 1 interface

Any attribute can be uniform or per-element

```
use '@use-gpu/wgsl/use/types'::{ SolidVertex };
use '@use-gpu/wgsl/use/view'::{ getViewResolution, worl                                    applyZBias };
use '@use-gpu/wgsl/geometry/quad'::{ getQuadUV };

@optional @link fn getPosition(i: u32) -> vec4<f32> { return vec4<f32>(0.0, 0.0, 0.0, 1.0); };
@optional @link fn getScissor(i: u32) -> vec4<f32> { return vec4<f3

@optional @link fn getRectangle(i: u32) -> vec4<f32> { return vec4<
@optional @link fn getColor(i: u32) -> vec4<f32> { return vec4<f32>
@optional @link fn getDepth(i: u32) -> f32 { return 0.0; };
@optional @link fn getZBias(i: u32) -> f32 { return 0.0; };
@optional @link fn getUV(i: u32) -> vec4<f32> { return vec4<f32>(0.
@optional @link fn getST(i: u32) -> vec4<f32> { return vec4<f32>(0.5, 0.5, 0.0, 0.0); };

@optional @link fn getPointCount() -> f32 { return 1.0;

@export fn getQuadVertex(vertexIndex: u32, elementIndex: u32) -> SolidVertex {

  var position = getPosition(elementIndex);
  var scissor = getScissor(elementIndex);
  var rectangle = getRectangle(elementIndex);
  var color = getColor(elementIndex);
  var depth = getDepth(elementIndex);
  var rectangleUV = getUV(elementIndex);
  var st4 = getST(elementIndex);
```

Could potentially be replaced with **1 struct** but what about **defaults** and **spread**?

Default value = fn body

Is a **getVertex(...)**

8 storage buffers max?

Aggregation

8 storage buffers max

Aggregation

8 storage buffers max

Group by **spread policy**

Aggregation

8 storage buffers max

Group by **spread policy**

```
40  export const SHAPE_SCHEMA = expandArrays({
41    ids:        {format: 'u32', single: 'id'},
42    lookups:    {format: 'u32', single: 'lookup'},
43    colors:     {format: 'vec4<f32>', single: 'color'},
44    zBiases:    {format: 'f32', single: 'zBias'},
45  });
46
47  export const MATRIX_SCHEMA = expandArrays({
48    matrices: {format: 'mat4x4<f32>', single: 'matrix', ref: true},
49    normalMatrices: {format: 'mat3x3<f32>', single: 'normalMatrix', ref: true},
50  });
51
52  export const POINT_SCHEMA = {
53    ...SHAPE_SCHEMA,
54    ...MATRIX_SCHEMA,
55    ...expandArrays({
56      positions: {format: 'vec4<f32>', single: 'position'},
57      sizes:     {format: 'f32', single: 'size'},
58      depths:    {format: 'f32', single: 'depth'},
59    }),
60  };
61
62  export const LINE_SEGMENTS_SCHEMA = expandArrays({
63    segments:  {format: 'i8', unwelded: true},
64  });
65
```

Aggregation

8 storage buffers max

Group by **spread policy**

```
40  export const SHAPE_SCHEMA = expandArrays({
41    ids:         {format: 'u32', single: 'id'},
42    lookups:     {format: 'u32', single: 'lookup'},
43    colors:      {format: 'vec4<f32>', single: 'color'},
44    zBiases:     {format: 'f32', single: 'zBias'},
45  });
46
47  export const MATRIX_SCHEMA = expandArrays({
48    matrices: {format: 'mat4x4<f32>', single: 'matrix', ref: true},
49    normalMatrices: {format: 'mat3x3<f32>', single: 'normalMatrix', ref: true},
50  });
51
52  export const POINT_SCHEMA = {
53    ...SHAPE_SCHEMA,
54    ...MATRIX_SCHEMA,
55    ...expandArrays({
56      positions: {format: 'vec4<f32>', single: 'position'},
57      sizes:     {format: 'f32', single: 'size'},
58      depths:    {format: 'f32', single: 'depth'},
59    }),
60  };
61
62  export const LINE_SEGMENTS_SCHEMA = expandArrays({
63    segments:  {format: 'i8', unwelded: true},
64  });
65
```

Data driven schema

Aggregation

8 storage buffers max

Group by **spread policy**

```
40  export const SHAPE_SCHEMA = expandArrays({
41    ids:         {format: 'u32', single: 'id'},
42    lookups:     {format: 'u32', single: 'lookup'},
43    colors:      {format: 'vec4<f32>', single: 'color'},
44    zBiases:     {format: 'f32', single: 'zBias'},
45  });
46
47  export const MATRIX_SCHEMA = expandArrays({
48    matrices: {format: 'mat4x4<f32>', single: 'matrix', ref: true},
49    normalMatrices: {format: 'mat3x3<f32>', single: 'normalMatrix', ref: true},
50  });
51
52  export const POINT_SCHEMA = {
53    ...SHAPE_SCHEMA,
54    ...MATRIX_SCHEMA,
55    ...expandArrays({
56      positions: {format: 'vec4<f32>', single: 'position'},
57      sizes:     {format: 'f32', single: 'size'},
58      depths:    {format: 'f32', single: 'depth'},
59    }),
60  };
61
62  export const LINE_SEGMENTS_SCHEMA = expandArrays({
63    segments:  {format: 'i8', unwelded: true},
64  });
65
```

Data driven schema

Plural vs singular convention

Aggregation

8 storage buffers max

Group by **spread policy**

```
40  export const SHAPE_SCHEMA = expandArrays({
41    ids:         {format: 'u32', single: 'id'},
42    lookups:     {format: 'u32', single: 'lookup'},
43    colors:      {format: 'vec4<f32>', single: 'color'},
44    zBiases:     {format: 'f32', single: 'zBias'},
45  });
46
47  export const MATRIX_SCHEMA = expandArrays({
48    matrices: {format: 'mat4x4<f32>', single: 'matrix', ref: true},
49    normalMatrices: {format: 'mat3x3<f32>', single: 'normalMatrix', ref: true},
50  });
51
52  export const POINT_SCHEMA = {
53    ...SHAPE_SCHEMA,
54    ...MATRIX_SCHEMA,
55    ...expandArrays({
56      positions: {format: 'vec4<f32>', single: 'position'},
57      sizes:     {format: 'f32', single: 'size'},
58      depths:    {format: 'f32', single: 'depth'},
59    }),
60  };
61
62  export const LINE_SEGMENTS_SCHEMA = expandArrays({
63    segments:  {format: 'i8', unwelded: true},
64  });
65
```

Data driven schema

Ref = volatile value

Plural vs singular convention

Aggregation

8 storage buffers max

Group by **spread policy**

```
40  export const SHAPE_SCHEMA = expandArrays({
41    ids:        {format: 'u32', single: 'id'},
42    lookups:    {format: 'u32', single: 'lookup'},
43    colors:     {format: 'vec4<f32>', single: 'color'},
44    zBiases:    {format: 'f32', single: 'zBias'},
45  });
46
47  export const MATRIX_SCHEMA = expandArrays({
48    matrices: {format: 'mat4x4<f32>', single: 'matrix', ref: true},
49    normalMatrices: {format: 'mat3x3<f32>', single: 'normalMatrix', ref: true},
50  });
51
52  export const POINT_SCHEMA = {
53    ...SHAPE_SCHEMA,
54    ...MATRIX_SCHEMA,
55    ...expandArrays({
56      positions: {format: 'vec4<f32>', single: 'position'},
57      sizes:     {format: 'f32', single: 'size'},
58      depths:    {format: 'f32', single: 'depth'},
59    }),
60  };
61
62  export const LINE_SEGMENTS_SCHEMA = expandArrays({
63    segments:  {format: 'i8', unwelded: true},
64  });
65
```

Data driven schema

Ref = volatile value

Plural vs singular convention

Polyfill **`i/u 8/16`** and unwelded attributes

Aggregation

8 storage buffers max

Group by **spread policy**

Aggregation

8 storage buffers max

Group by **spread policy**

```
42
43  //// @link @struct getPositionsAnchorsTrims _06_
44
45  struct _06_getPositionsAnchorsTrims {
46    positions: vec4<f32>,
47    anchors: vec4<u32>,
48    trims: vec4<u32>,
49  };
50
51  //// @link @struct getColorsWidthsSizesDepths _07_
52
53  struct _07_getColorsWidthsSizesDepths {
54    colors: vec4<f32>,
55    widths: f32,
56    sizes: f32,
57    depths: f32,
58  };
59
```

Aggregation

8 storage buffers max

Group by **spread policy**

```
43  //// @link @struct getPositionsAnchorsTrims _06_
44
45  struct _06_getPositionsAnchorsTrims {
46    positions: vec4<f32>,
47    anchors: vec4<u32>,
48    trims: vec4<u32>,
49  };
50
51  //// @link @struct getColorsWidthsSizesDepths _07_
52
53  struct _07_getColorsWidthsSizesDepths {
54    colors: vec4<f32>,
55    widths: f32,
56    sizes: f32,
57    depths: f32,
58  };
```

Per vertex

Aggregation

8 storage buffers max

Group by **spread policy**

```
43  //// @link @struct getPositionsAnchorsTrims _06_
44
45  struct _06_getPositionsAnchorsTrims {
46    positions: vec4<f32>,
47    anchors: vec4<u32>,
48    trims: vec4<u32>,
49  };
50
51  //// @link @struct getColorsWidthsSizesDepths _07_
52
53  struct _07_getColorsWidthsSizesDepths {
54    colors: vec4<f32>,
55    widths: f32,
56    sizes: f32,
57    depths: f32,
58  };
```

Per vertex

Per instance

Aggregation

8 storage buffers max

Group by **spread policy**

```
60  //// @link @access [getPositionsAnchorsTrims] [om0mr1wjq8] _08_
61
62  @group(1) @binding(1) var<storage> _08_getPositionsAnchorsTrims: array<_06_getPositionsAnchorsT
63

76
77  //// @link @access [getColorsWidthsSizesDepths] [jkrqdajhdl] _0a_
78
79  @group(1) @binding(4) var<storage> _0a_getColorsWidthsSizesDepths: array<_07_getColorsWidthsSiz
80
```

Aggregation

8 storage buffers max

Group by **spread policy**

One **storage buffer** per spread policy

```
60  //// @link @access [getPositionsAnchorsTrims] [om0mr1wjq8] _08_
61
62  @group(1) @binding(1) var<storage> _08_getPositionsAnchorsTrims: array<_06_getPositionsAnchorsT
63
76
77  //// @link @access [getColorsWidthsSizesDepths] [jkrqdajhdl] _0a_
78
79  @group(1) @binding(4) var<storage> _0a_getColorsWidthsSizesDepths: array<_07_getColorsWidthsSiz
80
```

Aggregation

8 storage buffers max

Group by **spread policy**

One **storage buffer** per spread policy

N attributes

```
60  //// @link @access [getPositionsAnchorsTrims] [om0mr1wjq8] _08_
61
62  @group(1) @binding(1) var<storage> _08_getPositionsAnchorsTrims: array<_06_getPositionsAnchorsT
63
76
77  //// @link @access [getColorsWidthsSizesDepths] [jkrqdajhdl] _0a_
78
79  @group(1) @binding(4) var<storage> _0a_getColorsWidthsSizesDepths: array<_07_getColorsWidthsSiz
80
```

Aggregation

8 storage buffers max

Group by **spread policy**

One **storage buffer** per spread policy

N attributes

```
60  //// @link @access [getPositionsAnchorsTrims] [om0mr1wjq8] _08_
61
62  @group(1) @binding(1) var<storage> _08_getPositionsAnchorsTrims: array<_06_getPositionsAnchorsT
63
76
77  //// @link @access [getColorsWidthsSizesDepths] [jkrqdajhdl] _0a_
78
79  @group(1) @binding(4) var<storage> _0a_getColorsWidthsSizesDepths: array<_07_getColorsWidthsSiz
80
```

Copy data into aggregated **binary buffers**

Aggregation

8 storage buffers max

Group by **spread policy**

```
80
81  //// @link @explode _0b_
82
83  fn _0b_positions(i: u32) → vec4<f32> {
84    return _08_getPositionsAnchorsTrims[i].positions;
85  }
86  fn _0b_anchors(i: u32) → vec4<u32> {
87    return _08_getPositionsAnchorsTrims[i].anchors;
88  }
89  fn _0b_trims(i: u32) → vec4<u32> {
90    return _08_getPositionsAnchorsTrims[i].trims;
91  }
92
93  //// @link @explode _0c_
94
95  fn _0c_colors(i: u32) → vec4<f32> {
96    return _0a_getColorsWidthsSizesDepths[i].colors;
97  }
98  fn _0c_widths(i: u32) → f32 {
99    return _0a_getColorsWidthsSizesDepths[i].widths;
100 }
101 fn _0c_sizes(i: u32) → f32 {
```

Aggregation

8 storage buffers max

Group by **spread policy**

```
80
81  //// @link @explode _0b_
82
83  fn _0b_positions(i: u32) → vec4<f32> {
84    return _08_getPositionsAnchorsTrims[i].positions;
85  }
86  fn _0b_anchors(i: u32) → vec4<u32> {
87    return _08_getPositionsAnchorsTrims[i].anchors;
88  }
89  fn _0b_trims(i: u32) → vec4<u32> {
90    return _08_getPositionsAnchorsTrims[i].trims;
91  }
92
93  //// @link @explode _0c_
94
95  fn _0c_colors(i: u32) → vec4<f32> {
96    return _0a_getColorsWidthsSizesDepths[i].colors;
97  }
98  fn _0c_widths(i: u32) → f32 {
99    return _0a_getColorsWidthsSizesDepths[i].widths;
100 }
101 fn _0c_sizes(i: u32) → f32 {
```

Explode into accessor functions

Aggregation

8 storage buffers max

Group by **spread policy**

```
141  //// @link @instanced [instances / colors widths sizes depths] _0g_
142
143  var<private> _0g_a: vec4<f32>;
144  var<private> _0g_b: f32;
145  var<private> _0g_c: f32;
146  var<private> _0g_d: f32;
147  fn _0g_loadIndex(index: u32) {
148    let mapped = _04_instances(index);
149    _0g_a = _0c_colors(mapped);
150    _0g_b = _0c_widths(mapped);
151    _0g_c = _0c_sizes(mapped);
152    _0g_d = _0c_depths(mapped);
153  }
154
155  fn _0g_colors(index: u32) → vec4<f32> { return _0g_a; }
156  fn _0g_widths(index: u32) → f32 { return _0g_b; }
157  fn _0g_sizes(index: u32) → f32 { return _0g_c; }
158  fn _0g_depths(index: u32) → f32 { return _0g_d; }
159
```

Aggregation

8 storage buffers max

Group by **spread policy**

```
141  //// @link @instanced [instances / colors widths sizes depths] _0g_
142
143  var<private> _0g_a: vec4<f32>;
144  var<private> _0g_b: f32;
145  var<private> _0g_c: f32;
146  var<private> _0g_d: f32;
147  fn _0g_loadIndex(index: u32) {
148    let mapped = _04_instances(index);
149    _0g_a = _0c_colors(mapped);
150    _0g_b = _0c_widths(mapped);
151    _0g_c = _0c_sizes(mapped);
152    _0g_d = _0c_depths(mapped);
153  }
154
155  fn _0g_colors(index: u32) → vec4<f32> { return _0g_a; }
156  fn _0g_widths(index: u32) → f32 { return _0g_b; }
157  fn _0g_sizes(index: u32) → f32 { return _0g_c; }
158  fn _0g_depths(index: u32) → f32 { return _0g_d; }
159
```

Replicate per-instance semantics in code

Aggregation

8 storage buffers max

Group by **spread policy**

```
141  //// @link @instanced [instances / colors widths sizes depths] _0g_
142
143  var<private> _0g_a: vec4<f32>;
144  var<private> _0g_b: f32;
145  var<private> _0g_c: f32;
146  var<private> _0g_d: f32;
147  fn _0g_loadIndex(index: u32) {
148    let mapped = _04_instances(index);
149    _0g_a = _0c_colors(mapped);
150    _0g_b = _0c_widths(mapped);
151    _0g_c = _0c_sizes(mapped);
152    _0g_d = _0c_depths(mapped);
153  }
154
155  fn _0g_colors(index: u32) → vec4<f32> { return _0g_a; }
156  fn _0g_widths(index: u32) → f32 { return _0g_b; }
157  fn _0g_sizes(index: u32) → f32 { return _0g_c; }
158  fn _0g_depths(index: u32) → f32 { return _0g_d; }
159
```

Replicate per-instance semantics in code

Explode into accessor functions

Aggregation

8 storage buffers max

Group by **spread policy**

```
141  //// @link @instanced [instances / colors widths sizes depths] _0g_
142
143  var<private> _0g_a: vec4<f32>;
144  var<private> _0g_b: f32;
145  var<private> _0g_c: f32;
146  var<private> _0g_d: f32;
147  fn _0g_loadIndex(index: u32) {
148    let mapped = _04_instances(index);
149    _0g_a = _0c_colors(mapped);
150    _0g_b = _0c_widths(mapped);
151    _0g_c = _0c_sizes(mapped);
152    _0g_d = _0c_depths(mapped);
153  }
154
155  fn _0g_colors(index: u32) → vec4<f32> { return _0g_a; }
156  fn _0g_widths(index: u32) → f32 { return _0g_b; }
157  fn _0g_sizes(index: u32) → f32 { return _0g_c; }
158  fn _0g_depths(index: u32) → f32 { return _0g_d; }
159
```

Replicate per-instance semantics in code

Remapping indices = make hardware obsolete

Explode into accessor functions

Aggregation

8 storage buffers max

Group by **spread policy**

```
141  //// @link @instanced [instances / colors widths sizes depths] _0g_
142
143  var<private> _0g_a: vec4<f32>;
144  var<private> _0g_b: f32;
145  var<private> _0g_c: f32;
146  var<private> _0g_d: f32;
147  fn _0g_loadIndex(index: u32) {
148    let mapped = _04_instances(index);
149    _0g_a = _0c_colors(mapped);
150    _0g_b = _0c_widths(mapped);
151    _0g_c = _0c_sizes(mapped);
152    _0g_d = _0c_depths(mapped);
153  }
154
155  fn _0g_colors(index: u32) → vec4<f32> { return _0g_a; }
156  fn _0g_widths(index: u32) → f32 { return _0g_b; }
157  fn _0g_sizes(index: u32) → f32 { return _0g_c; }
158  fn _0g_depths(index: u32) → f32 { return _0g_d; }
159
```

Replicate per-instance semantics in code

Remapping indices = make hardware obsolete

** It's not actually hardware IRL anyway*

Explode into accessor functions

Aggregation

8 storage buffers max

Group by **spread policy**

```
74  @group(1) @binding(0) var<storage> _VT_1_getSegment8to32Storage: array<u32>;
75
76  fn _VT_1_getSegment8to32(a: u32) → u32 {
77    return _VT_1_getSegment8to32Storage[a];
78  }
79
80  fn _VT_1_getSegment(i: u32) → i32 {
81    let b2 = i >> 2u;
82    let f4 = i & 3u;
83
84    let word = u32(_VT_1_getSegment8to32(b2));
85    var v: i32 = i32((word >> (f4 << 3u)) & 0×FFu);
86    return v;
87  }
```

Aggregation

8 storage buffers max

Group by **spread policy**

```
74  @group(1) @binding(0) var<storage> _VT_1_getSegment8to32Storage: array<u32>;
75
76  fn _VT_1_getSegment8to32(a: u32) → u32 {
77    return _VT_1_getSegment8to32Storage[a];
78  }
79
80  fn _VT_1_getSegment(i: u32) → i32 {
81    let b2 = i >> 2u;
82    let f4 = i & 3u;
83
84    let word = u32(_VT_1_getSegment8to32(b2));
85    var v: i32 = i32((word >> (f4 << 3u)) & 0×FFu);
86    return v;
87  }
```

Polyfill **i8** as **array<u32>**

Compute Orchestration

```
const makeBlurShader = (i: number, pass: number) =>
  getShader(pmremBlur, [
    mappings[i],
    pass ? mappings[i] : mappings[i - 1],
    dsigmas[i],
    radii[i],
    scratchIn,
    scratchOut,
    target,
  ], {
    BLUR_PASS: pass,
    SIGMA_CUTOFF,
    MAX_SAMPLES,
  });

const makeDispatch = (shader: ShaderModule, i: number) =>
  use(Dispatch, {
    shader,
    size: [sizes[i], sizes[i]],
    group: [8, 8],
    onDispatch: scratch.swap,
  });
```

Compute Orchestration

```
const makeBlurShader = (i: number, pass: number) =>
  getShader(pmremBlur, [
    mappings[i],
    pass ? mappings[i] : mappings[i - 1],
    dsigmas[i],
    radii[i],
    scratchIn,
    scratchOut,
    target,
  ], {
    BLUR_PASS: pass,
    SIGMA_CUTOFF,
    MAX_SAMPLES,
  });

const makeDispatch = (shader: ShaderModule, i: number) =>
  use(Dispatch, {
    shader,
    size: [sizes[i], sizes[i]],
    group: [8, 8],
    onDispatch: scratch.swap,
  });
```

getShader =
non-reactive **useShader**

Compute Orchestration

```
const makeBlurShader = (i: number, pass: number) =>
  getShader(pmremBlur, [
    mappings[i],
    pass ? mappings[i] : mappings[i - 1],
    dsigmas[i],
    radii[i],
    scratchIn,
    scratchOut,
    target,
  ], {
    BLUR_PASS: pass,
    SIGMA_CUTOFF,
    MAX_SAMPLES,
  });

const makeDispatch = (shader: ShaderModule, i: number) =>
  use(Dispatch, {
    shader,
    size: [sizes[i], sizes[i]],
    group: [8, 8],
    onDispatch: scratch.swap,
  });
```

getShader =
non-reactive **useShader**

The entire dispatch is
prepared in one
big **useMemo(...)**

Compute Orchestration

```
const makeBlurShader = (i: number, pass: number) =>
  getShader(pmremBlur, [
    mappings[i],
    pass ? mappings[i] : mappings[i - 1],
    dsigmas[i],
    radii[i],
    scratchIn,
    scratchOut,
    target,
  ], {
    BLUR_PASS: pass,
    SIGMA_CUTOFF,
    MAX_SAMPLES,
  });

const makeDispatch = (shader: ShaderModule, i: number) =>
  use(Dispatch, {
    shader,
    size: [sizes[i], sizes[i]],
    group: [8, 8],
    onDispatch: scratch.swap,
  });
```

getShader =
non-reactive **useShader**

The entire dispatch is
prepared in one
big **useMemo(...)**

Compute dispatch
added to WebGPU queue

Compute Orchestration

```
const makeBlurShader = (i: number, pass: number) =>
  getShader(pmremBlur, [
    mappings[i],
    pass ? mappings[i] : mappings[i - 1],
    dsigmas[i],
    radii[i],
    scratchIn,
    scratchOut,
    target,
  ], {
    BLUR_PASS: pass,
    SIGMA_CUTOFF,
    MAX_SAMPLES,
  });

const makeDispatch = (shader: ShaderModule, i: number)
  use(Dispatch, {
    shader,
    size: [sizes[i], sizes[i]],
    group: [8, 8],
    onDispatch: scratch.swap,
  });
```

getShader =
non-reactive **useShader**

The entire dispatch is
prepared in one
big **useMemo(...)**

use(Foo) = <Foo />

Compute dispatch
added to WebGPU queue

Compute Orchestration

```
const makeBlurShader = (i: number, pass: number) =>
  getShader(pmremBlur, [
    mappings[i],
    pass ? mappings[i] : mappings[i - 1],
    dsigmas[i],
    radii[i],
    scratchIn,
    scratchOut,
    target,
  ], {
    BLUR_PASS: pass,
    SIGMA_CUTOFF,
    MAX_SAMPLES,
  });

const makeDispatch = (shader: ShaderModule, i: number)
  use(Dispatch, {
    shader,
    size: [sizes[i], sizes[i]],
    group: [8, 8],
    onDispatch: scratch.swap,
  });
```

getShader =
non-reactive **useShader**

The entire dispatch is
prepared in one
big **useMemo(...)**

use(Foo) = <Foo />

Compute dispatch
added to WebGPU queue

Similar for
`use(DrawCall, { ... })`

Data Structures

Module & Bundle

Module & Bundle

string

```
fn signNotZero(xy: vec2<f32>) -> vec2<f32> {
  let s = sign(xy);
  return select(s, vec2<f32>(1.0, 1.0), s == vec2<f32>(0.0));
}

/** Assumes that v is a unit vector. The result is an octahedral vector on the [-1, +1] square. */
@export fn encodeOctahedral(v: vec3<f32>) -> vec2<f32> {
  let l1norm = abs(v.x) + abs(v.y) + abs(v.z);
  var result = v.xy * (1.0 / l1norm);
  if (v.z < 0.0) {
    result = (1.0 - abs(result.yx)) * signNotZero(result.xy);
  }
  return result;
}

/** Returns a unit vector. Argument o is an octahedral vector packed via encodeOctahedral,
    on the [-1, +1] square */
@export fn decodeOctahedral(o: vec2<f32>) -> vec3<f32> {
  var v = vec3<f32>(o.x, o.y, 1.0 - abs(o.x) - abs(o.y));
  if (v.z < 0.0) {
    v = vec3<f32>((1.0 - abs(v.yx)) * signNotZero(v.xy), v.z);
  }
  return normalize(v);
}
```

Module & Bundle

string

```
fn signNotZero(xy: vec2<f32>) -> vec2<f32> {
  let s = sign(xy);
  return select(s, vec2<f32>(1.0, 1.0), s == vec2<f32>(0.0));
}

/** Assumes that v is a unit vector. The result is an octahedral vector on the [-1, +1] square. */
@export fn encodeOctahedral(v: vec3<f32>) -> vec2<f32> {
  let l1norm = abs(v.x) + abs(v.y) + abs(v.z);
  var result = v.xy * (1.0 / l1norm);
  if (v.z < 0.0) {
    result = (1.0 - abs(result.yx)) * signNotZero(result.xy);
  }
  return result;
}

/** Returns a unit vector. Argument o is an octahedral vector packed via encodeOctahedral,
    on the [-1, +1] square */
@export fn decodeOctahedral(o: vec2<f32>) -> vec3<f32> {
  var v = vec3<f32>(o.x, o.y, 1.0 - abs(o.x) - abs(o.y));
  if (v.z < 0.0) {
    v = vec3<f32>((1.0 - abs(v.yx)) * signNotZero(v.xy), v.z);
  }
  return normalize(v);
}
```

Code = source of truth

Module & Bundle

string

```
fn signNotZero(xy: vec2<f32>) -> vec2<f32> {
  let s = sign(xy);
  return select(s, vec2<f32>(1.0, 1.0), s == vec2<f32>(0.0));
}

/** Assumes that v is a unit vector. The result is an octahedral vector on the [-1, +1] square. */
@export fn encodeOctahedral(v: vec3<f32>) -> vec2<f32> {
  let l1norm = abs(v.x) + abs(v.y) + abs(v.z);
  var result = v.xy * (1.0 / l1norm);
  if (v.z < 0.0) {
    result = (1.0 - abs(result.yx)) * signNotZero(result.xy);
  }
  return result;
}

/** Returns a unit vector. Argument o is an octahedral vector packed via encodeOctahedral,
    on the [-1, +1] square */
@export fn decodeOctahedral(o: vec2<f32>) -> vec3<f32> {
  var v = vec3<f32>(o.x, o.y, 1.0 - abs(o.x) - abs(o.y));
  if (v.z < 0.0) {
    v = vec3<f32>((1.0 - abs(v.yx)) * signNotZero(v.xy), v.z);
  }
  return normalize(v);
}
```

Name = bundler path

Code = source of truth

Module & Bundle

string

```
fn signNotZero(xy: vec2<f32>) -> vec2<f32> {
  let s = sign(xy);
  return select(s, vec2<f32>(1.0, 1.0), s == vec2<f32>(0.0));
}

/** Assumes that v is a unit vector. The result is an octahedral vector on the [-1, +1] square. */
@export fn encodeOctahedral(v: vec3<f32>) -> vec2<f32> {
  let l1norm = abs(v.x) + abs(v.y) + abs(v.z);
  var result = v.xy * (1.0 / l1norm);
  if (v.z < 0.0) {
    result = (1.0 - abs(result.yx)) * signNotZero(result.xy);
  }
  return result;
}

/** Returns a unit vector. Argument o is an octahedral vector packed via encodeOctahedral,
    on the [-1, +1] square */
@export fn decodeOctahedral(o: vec2<f32>) -> vec3<f32> {
  var v = vec3<f32>(o.x, o.y, 1.0 - abs(o.x) - abs(o.y));
  if (v.z < 0.0) {
    v = vec3<f32>((1.0 - abs(v.yx)) * signNotZero(v.xy), v.z);
  }
  return normalize(v);
}
```

Or inline **wgsl`...`**

Name = bundler path

Code = source of truth

Module & Bundle

string

```
fn signNotZero(xy: vec2<f32>) -> vec2<f32> {
  let s = sign(xy);
  return select(s, vec2<f32>(1.0, 1.0), s == vec2<f32>(0.0));
}

/** Assumes that v is a unit vector. The result is an octahedral vector on the [-1, +1] square. */
@export fn encodeOctahedral(v: vec3<f32>) -> vec2<f32> {
  let l1norm = abs(v.x) + abs(v.y) + abs(v.z);
  var result = v.xy * (1.0 / l1norm);
  if (v.z < 0.0) {
    result = (1.0 - abs(result.yx)) * signNotZero(result.xy);
  }
  return result;
}

/** Returns a unit vector. Argument o is an octahedral vector packed via encodeOctahedral,
    on the [-1, +1] square */
@export fn decodeOctahedral(o: vec2<f32>) -> vec3<f32> {
  var v = vec3<f32>(o.x, o.y, 1.0 - abs(o.x) - abs(o.y));
  if (v.z < 0.0) {
    v = vec3<f32>((1.0 - abs(v.yx)) * signNotZero(v.xy), v.z);
  }
  return normalize(v);
}
```

Or inline **`wgsl` ... `**

Name = bundler path

Code = source of truth

ParsedModule

```
- name: "..."
- code: "..."
- entry?: "..."
- table: { ... }
- tree: { ... }
- shake: { ... }
```

Module & Bundle

string

```
fn signNotZero(xy: vec2<f32>) -> vec2<f32> {
  let s = sign(xy);
  return select(s, vec2<f32>(1.0, 1.0), s == vec2<f32>(0.0));
}

/** Assumes that v is a unit vector. The result is an octahedral vector on the [-1, +1] square. */
@export fn encodeOctahedral(v: vec3<f32>) -> vec2<f32> {
  let l1norm = abs(v.x) + abs(v.y) + abs(v.z);
  var result = v.xy * (1.0 / l1norm);
  if (v.z < 0.0) {
    result = (1.0 - abs(result.yx)) * signNotZero(result.xy);
  }
  return result;
}

/** Returns a unit vector. Argument o is an octahedral vector packed via encodeOctahedral,
    on the [-1, +1] square */
@export fn decodeOctahedral(o: vec2<f32>) -> vec3<f32> {
  var v = vec3<f32>(o.x, o.y, 1.0 - abs(o.x) - abs(o.y));
  if (v.z < 0.0) {
    v = vec3<f32>((1.0 - abs(v.yx)) * signNotZero(v.xy), v.z);
  }
  return normalize(v);
}
```

Or inline **wgsl`...`**

Name = bundler path

Code = source of truth

ParsedModule

```
- name: "..."
- code: "..."
- entry?: "..."
- table: { ... }
- tree: { ... }
- shake: { ... }
```

Symbol Table +
Sparse AST + Tree shaking

Module & Bundle

string

```
fn signNotZero(xy: vec2<f32>) -> vec2<f32> {
  let s = sign(xy);
  return select(s, vec2<f32>(1.0, 1.0), s == vec2<f32>(0.0));
}

/** Assumes that v is a unit vector. The result is an octahedral vector on the [-1, +1] square. */
@export fn encodeOctahedral(v: vec3<f32>) -> vec2<f32> {
  let l1norm = abs(v.x) + abs(v.y) + abs(v.z);
  var result = v.xy * (1.0 / l1norm);
  if (v.z < 0.0) {
    result = (1.0 - abs(result.yx)) * signNotZero(result.xy);
  }
  return result;
}

/** Returns a unit vector. Argument o is an octahedral vector packed via encodeOctahedral,
    on the [-1, +1] square */
@export fn decodeOctahedral(o: vec2<f32>) -> vec3<f32> {
  var v = vec3<f32>(o.x, o.y, 1.0 - abs(o.x) - abs(o.y));
  if (v.z < 0.0) {
    v = vec3<f32>((1.0 - abs(v.yx)) * signNotZero(v.xy), v.z);
  }
  return normalize(v);
}
```

Or inline `wgsl` ... `

Name = bundler path

Code = source of truth

ParsedModule

- `name: " ... "`
- `code: " ... "`
- `entry?: " ... "`
- `table: { ... }`
- `tree: { ... }`
- `shake: { ... }`

Symbol Table +
Sparse AST + Tree shaking

ParsedBundle

- `module: { ... }`
- `entry?: " ... "`
- `libs: { ... }`
- `links: { ... }`

Module & Bundle

string

```wgsl
fn signNotZero(xy: vec2<f32>) -> vec2<f32> {
  let s = sign(xy);
  return select(s, vec2<f32>(1.0, 1.0), s == vec2<f32>(0.0));
}

/** Assumes that v is a unit vector. The result is an octahedral vector on the [-1, +1] square. */
@export fn encodeOctahedral(v: vec3<f32>) -> vec2<f32> {
  let l1norm = abs(v.x) + abs(v.y) + abs(v.z);
  var result = v.xy * (1.0 / l1norm);
  if (v.z < 0.0) {
    result = (1.0 - abs(result.yx)) * signNotZero(result.xy);
  }
  return result;
}

/** Returns a unit vector. Argument o is an octahedral vector packed via encodeOctahedral,
    on the [-1, +1] square */
@export fn decodeOctahedral(o: vec2<f32>) -> vec3<f32> {
  var v = vec3<f32>(o.x, o.y, 1.0 - abs(o.x) - abs(o.y));
  if (v.z < 0.0) {
    v = vec3<f32>((1.0 - abs(v.yx)) * signNotZero(v.xy), v.z);
  }
  return normalize(v);
}
```

Or inline **`wgsl` ... `**

Name = bundler path

Code = source of truth

ParsedModule

- **name: "..."**
- **code: "..."**
- **entry?: "..."**
- **table: { ... }**
- **tree: { ... }**
- **shake: { ... }**

Symbol Table +
Sparse AST + Tree shaking

ParsedBundle

- **module: { ... }**
- **entry?: "..."**
- **libs: { ... }**
- **links: { ... }**

Module + Libs + Links
(recursive)

Module & Bundle

string

```
fn signNotZero(xy: vec2<f32>) -> vec2<f32> {
  let s = sign(xy);
  return select(s, vec2<f32>(1.0, 1.0), s == vec2<f32>(0.0));
}

/** Assumes that v is a unit vector. The result is an octahedral vector on the [-1, +1] square. */
@export fn encodeOctahedral(v: vec3<f32>) -> vec2<f32> {
  let l1norm = abs(v.x) + abs(v.y) + abs(v.z);
  var result = v.xy * (1.0 / l1norm);
  if (v.z < 0.0) {
    result = (1.0 - abs(result.yx)) * signNotZero(result.xy);
  }
  return result;
}

/** Returns a unit vector. Argument o is an octahedral vector packed via encodeOctahedral,
    on the [-1, +1] square */
@export fn decodeOctahedral(o: vec2<f32>) -> vec3<f32> {
  var v = vec3<f32>(o.x, o.y, 1.0 - abs(o.x) - abs(o.y));
  if (v.z < 0.0) {
    v = vec3<f32>((1.0 - abs(v.yx)) * signNotZero(v.xy), v.z);
  }
  return normalize(v);
}
```

Or inline **`wgsl` ... `**

Name = bundler path

Code = source of truth

ParsedModule

- **name: "..."**
- **code: "..."**
- **entry?: "..."**
- **table: { ... }**
- **tree: { ... }**
- **shake: { ... }**

Symbol Table +
Sparse AST + Tree shaking

ParsedBundle

- **module: { ... }**
- **entry?: "..."**
- **libs: { ... }**
- **links: { ... }**

Linking is **lazy**
Bind is cheap

Module + Libs + Links
(recursive)

Module & Bundle

string

```
fn signNotZero(xy: vec2<f32>) -> vec2<f32> {
  let s = sign(xy);
  return select(s, vec2<f32>(1.0, 1.0), s == vec2<f32>(0.0));
}

/** Assumes that v is a unit vector. The result is an octahedral vector on the [-1, +1] square. */
@export fn encodeOctahedral(v: vec3<f32>) -> vec2<f32> {
  let l1norm = abs(v.x) + abs(v.y) + abs(v.z);
  var result = v.xy * (1.0 / l1norm);
  if (v.z < 0.0) {
    result = (1.0 - abs(result.yx)) * signNotZero(result.xy);
  }
  return result;
}

/** Returns a unit vector. Argument o is an octahedral vector packed via encodeOctahedral,
    on the [-1, +1] square */
@export fn decodeOctahedral(o: vec2<f32>) -> vec3<f32> {
  var v = vec3<f32>(o.x, o.y, 1.0 - abs(o.x) - abs(o.y));
  if (v.z < 0.0) {
    v = vec3<f32>((1.0 - abs(v.yx)) * signNotZero(v.xy), v.z);
  }
  return normalize(v);
}
```

ParsedModule

```
- name: "..."
- code: "..."
- entry?: "..."
- table: {...}
- tree: {...}
- shake: {...}
- hash: 0×1234
- key?: 0×5678
```

ParsedBundle

```
- module: {...}
- entry?: "..."
- libs: {...}
- links: {...}
- hash: 0×abcd
- key?: 0×fecb
```

Module & Bundle

Structural **hash**
+ instance **key**

string

```
fn signNotZero(xy: vec2<f32>) -> vec2<f32> {
  let s = sign(xy);
  return select(s, vec2<f32>(1.0, 1.0), s == vec2<f32>(0.0));
}

/** Assumes that v is a unit vector. The result is an octahedral vector on the [-1, +1] square. */
@export fn encodeOctahedral(v: vec3<f32>) -> vec2<f32> {
  let l1norm = abs(v.x) + abs(v.y) + abs(v.z);
  var result = v.xy * (1.0 / l1norm);
  if (v.z < 0.0) {
    result = (1.0 - abs(result.yx)) * signNotZero(result.xy);
  }
  return result;
}

/** Returns a unit vector. Argument o is an octahedral vector packed via encodeOctahedral,
    on the [-1, +1] square */
@export fn decodeOctahedral(o: vec2<f32>) -> vec3<f32> {
  var v = vec3<f32>(o.x, o.y, 1.0 - abs(o.x) - abs(o.y));
  if (v.z < 0.0) {
    v = vec3<f32>((1.0 - abs(v.yx)) * signNotZero(v.xy), v.z);
  }
  return normalize(v);
}
```

ParsedModule

- name: "..."
- code: "..."
- entry?: "..."
- table: { ... }
- tree: { ... }
- shake: { ... }
- hash: 0×1234
- key?: 0×5678

ParsedBundle

- module: { ... }
- entry?: "..."
- libs: { ... }
- links: { ... }
- hash: 0×abcd
- key?: 0×fecb

Module & Bundle

Structural **hash**
+ instance **key**

string

```
fn signNotZero(xy: vec2<f32>) -> vec2<f32> {
  let s = sign(xy);
  return select(s, vec2<f32>(1.0, 1.0), s == vec2<f32>(0.0));
}

/** Assumes that v is a unit vector. The result is an octahedral vector on the [-1, +1] square. */
@export fn encodeOctahedral(v: vec3<f32>) -> vec2<f32> {
  let l1norm = abs(v.x) + abs(v.y) + abs(v.z);
  var result = v.xy * (1.0 / l1norm);
  if (v.z < 0.0) {
    result = (1.0 - abs(result.yx)) * signNotZero(result.xy);
  }
  return result;
}

/** Returns a unit vector. Argument o is an octahedral vector packed via encodeOctahedral,
    on the [-1, +1] square */
@export fn decodeOctahedral(o: vec2<f32>) -> vec3<f32> {
  var v = vec3<f32>(o.x, o.y, 1.0 - abs(o.x) - abs(o.y));
  if (v.z < 0.0) {
    v = vec3<f32>((1.0 - abs(v.yx)) * signNotZero(v.xy), v.z);
  }
  return normalize(v);
}
```

ParsedModule

```
- name: "..."
- code: "..."
- entry?: "..."
- table: { ... }
- tree: { ... }
- shake: { ... }
- hash: 0×1234
- key?: 0×5678
```

ParsedBundle

```
- module: { ... }
- entry?: "..."
- libs: { ... }
- links: { ... }
- hash: 0×abcd
- key?: 0×fecb
```

Bind **code** = change **hash**

Module & Bundle

Structural **hash**
+ instance **key**

string

```
fn signNotZero(xy: vec2<f32>) -> vec2<f32> {
  let s = sign(xy);
  return select(s, vec2<f32>(1.0, 1.0), s == vec2<f32>(0.0));
}

/** Assumes that v is a unit vector. The result is an octahedral vector on the [-1, +1] square. */
@export fn encodeOctahedral(v: vec3<f32>) -> vec2<f32> {
  let l1norm = abs(v.x) + abs(v.y) + abs(v.z);
  var result = v.xy * (1.0 / l1norm);
  if (v.z < 0.0) {
    result = (1.0 - abs(result.yx)) * signNotZero(result.xy);
  }
  return result;
}

/** Returns a unit vector. Argument o is an octahedral vector packed via encodeOctahedral,
    on the [-1, +1] square */
@export fn decodeOctahedral(o: vec2<f32>) -> vec3<f32> {
  var v = vec3<f32>(o.x, o.y, 1.0 - abs(o.x) - abs(o.y));
  if (v.z < 0.0) {
    v = vec3<f32>((1.0 - abs(v.yx)) * signNotZero(v.xy), v.z);
  }
  return normalize(v);
}
```

ParsedModule

```
- name: "..."
- code: "..."
- entry?: "..."
- table: { ... }
- tree: { ... }
- shake: { ... }
- hash: 0×1234
- key?: 0×5678
```

ParsedBundle

```
- module: { ... }
- entry?: "..."
- libs: { ... }
- links: { ... }
- hash: 0×abcd
- key?: 0×fecb
```

Bind **code** = change **hash**

Bind **data** = change **key**

Module & Bundle

Structural **hash**
+ instance **key**

string

```
fn signNotZero(xy: vec2<f32>) -> vec2<f32> {
  let s = sign(xy);
  return select(s, vec2<f32>(1.0, 1.0), s == vec2<f32>(0.0));
}

/** Assumes that v is a unit vector. The result is an octahedral vector on the [-1, +1] square. */
@export fn encodeOctahedral(v: vec3<f32>) -> vec2<f32> {
  let l1norm = abs(v.x) + abs(v.y) + abs(v.z);
  var result = v.xy * (1.0 / l1norm);
  if (v.z < 0.0) {
    result = (1.0 - abs(result.yx)) * signNotZero(result.xy);
  }
  return result;
}

/** Returns a unit vector. Argument o is an octahedral vector packed via encodeOctahedral,
    on the [-1, +1] square */
@export fn decodeOctahedral(o: vec2<f32>) -> vec3<f32> {
  var v = vec3<f32>(o.x, o.y, 1.0 - abs(o.x) - abs(o.y));
  if (v.z < 0.0) {
    v = vec3<f32>((1.0 - abs(v.yx)) * signNotZero(v.xy), v.z);
  }
  return normalize(v);
}
```

ParsedModule

```
- name: "..."
- code: "..."
- entry?: "..."
- table: {...}
- tree: {...}
- shake: {...}
- hash: 0×1234
- key?: 0×5678
```

ParsedBundle

```
- module: {...}
- entry?: "..."
- libs: {...}
- links: {...}
- hash: 0×abcd
- key?: 0×fecb
```

Bind **code** = change **hash**

Bind **data** = change **key**

Unbound **hash** = **key**

Module & Bundle

Structural **hash**
+ instance **key**

string

```
fn signNotZero(xy: vec2<f32>) -> vec2<f32> {
  let s = sign(xy);
  return select(s, vec2<f32>(1.0, 1.0), s == vec2<f32>(0.0));
}

/** Assumes that v is a unit vector. The result is an octahedral vector on the [-1, +1] square. */
@export fn encodeOctahedral(v: vec3<f32>) -> vec2<f32> {
  let l1norm = abs(v.x) + abs(v.y) + abs(v.z);
  var result = v.xy * (1.0 / l1norm);
  if (v.z < 0.0) {
    result = (1.0 - abs(result.yx)) * signNotZero(result.xy);
  }
  return result;
}

/** Returns a unit vector. Argument o is an octahedral vector packed via encodeOctahedral,
    on the [-1, +1] square */
@export fn decodeOctahedral(o: vec2<f32>) -> vec3<f32> {
  var v = vec3<f32>(o.x, o.y, 1.0 - abs(o.x) - abs(o.y));
  if (v.z < 0.0) {
    v = vec3<f32>((1.0 - abs(v.yx)) * signNotZero(v.xy), v.z);
  }
  return normalize(v);
}
```

ParsedModule

```
- name: "..."
- code: "..."
- entry?: "..."
- table: { ... }
- tree: { ... }
- shake: { ... }
- hash: 0×1234
- key?: 0×5678
```

ParsedBundle

```
- module: { ... }
- entry?: "..."
- libs: { ... }
- links: { ... }
- hash: 0×abcd
- key?: 0×fecb
```

Change entry point
= change **hash**, not **key**

Bind **code** = change **hash**

Bind **data** = change **key**

Unbound **hash** = **key**

Virtual Module

string = lazy

```
  const access = readWrite ? 'storage, read_write' : 'storage';

  if (args === null) {
    return `@group(${set}) @binding(${binding}) var<${access}> ${ns}${name}: ${type};\n`;
  }

  const hasCast = needsCast(format, type);
  return (
`@group(${set}) @binding(${binding}) var<${access}> ${ns}${name}Storage: array<${format}>;

fn ${ns}${name}(${args.map((t, i) => `${arg(i)}: ${t}`).join(', ')}) -> ${type} {
  ${hasCast ? 'let v =' : 'return'} ${ns}${name}Storage[${args.length ? arg(0) : '0u'}];
${hasCast ? `  return ${makeSwizzle(format, type, 'v')};\n` : ''
}}
`\.
```

VirtualTable

```
- render: (namespace, ...) ⇒ string

- uniforms?: DataBinding<T>[],
- storages?: DataBinding<T>[],
- textures?: DataBinding<T>[],

- bindingBase?: number,
- volatileBase?: number,
- namespace?: string,
```

Virtual Module

string = lazy

```
  const access = readWrite ? 'storage, read_write' : 'storage';

  if (args === null) {
    return `@group(${set}) @binding(${binding}) var<${access}> ${ns}${name}: ${type};\n`;
  }

  const hasCast = needsCast(format, type);
  return (
`@group(${set}) @binding(${binding}) var<${access}> ${ns}${name}Storage: array<${format}>;

fn ${ns}${name}(${args.map((t, i) => `${arg(i)}: ${t}`).join(', ')}) -> ${type} {
  ${hasCast ? 'let v =' : 'return'} ${ns}${name}Storage[${args.length ? arg(0) : '0u'}];
${hasCast ? `  return ${makeSwizzle(format, type, 'v')};\n` : ''
}}
`\.
```

VirtualTable

- `render: (namespace, ...) ⇒ string`

- `uniforms?: DataBinding<T>[],`
- `storages?: DataBinding<T>[],`
- `textures?: DataBinding<T>[],`

- `bindingBase?: number,`
- `volatileBase?: number,`
- `namespace?: string,`

Render template

Virtual Module

string = lazy

```
  const access = readWrite / 'storage, read_write' : 'storage';

  if (args === null) {
    return `@group(${set}) @binding(${binding}) var<${access}> ${ns}${name}: ${type};\n`;
  }

  const hasCast = needsCast(format, type);
  return (
`@group(${set}) @binding(${binding}) var<${access}> ${ns}${name}Storage: array<${format}>;

fn ${ns}${name}(${args.map((t, i) => `${arg(i)}: ${t}`).join(', ')}) -> ${type} {
  ${hasCast ? 'let v =' : 'return'} ${ns}${name}Storage[${args.length ? arg(0) : '0u'}];
${hasCast ? `  return ${makeSwizzle(format, type, 'v')};\n` : ''
}}
`\.
```

VirtualTable

```
- render: (namespace, ...) ⇒ string
```

```
- uniforms?: DataBinding<T>[],
- storages?: DataBinding<T>[],
- textures?: DataBinding<T>[],
```

```
- bindingBase?: number,
- volatileBase?: number,
- namespace?: string,
```

Render template

Data bindings

Virtual Module

string = lazy

```
  const access = readWrite / 'storage, read_write' : 'storage';

  if (args === null) {
    return `@group(${set}) @binding(${binding}) var<${access}> ${ns}${name}: ${type};\n`;
  }

  const hasCast = needsCast(format, type);
  return (
`@group(${set}) @binding(${binding}) var<${access}> ${ns}${name}Storage: array<${format}>;

fn ${ns}${name}(${args.map((t, i) => `${arg(i)}: ${t}`).join(', ')}) -> ${type} {
  ${hasCast ? 'let v =' : 'return'} ${ns}${name}Storage[${args.length ? arg(0) : '0u'}];
${hasCast ? `  return ${makeSwizzle(format, type, 'v')};\n` : ''
}}
`\.
```

VirtualTable

```
- render: (namespace, ...) ⇒ string

- uniforms?: DataBinding<T>[],
- storages?: DataBinding<T>[],
- textures?: DataBinding<T>[],

- bindingBase?: number,
- volatileBase?: number,
- namespace?: string,
```

Render template

Data bindings

Resolved bindings at **link**-time

Virtual Module

string = lazy

```
  const access = readWrite ? 'storage, read_write' : 'storage';

  if (args === null) {
    return `@group(${set}) @binding(${binding}) var<${access}> ${ns}${name}: ${type};\n`;
  }

  const hasCast = needsCast(format, type);
  return (
`@group(${set}) @binding(${binding}) var<${access}> ${ns}${name}Storage: array<${format}>;

fn ${ns}${name}(${args.map((t, i) => `${arg(i)}: ${t}`).join(', ')}) -> ${type} {
  ${hasCast ? 'let v =' : 'return'} ${ns}${name}Storage[${args.length ? arg(0) : '0u'}];
${hasCast ? `  return ${makeSwizzle(format, type, 'v')};\n` : ''
}}
`\.
```

VirtualTable

```
- render: (namespace, ...) ⇒ string

- uniforms?: DataBinding<T>[],
- storages?: DataBinding<T>[],
- textures?: DataBinding<T>[],

- bindingBase?: number,
- volatileBase?: number,
- namespace?: string,
```

For a **vertex & fragment** pair

Render template

Data bindings

Resolved bindings at **link**-time

Virtual Module

string = lazy

```
  const access = readWrite ? 'storage, read_write' : 'storage';

  if (args === null) {
    return `@group(${set}) @binding(${binding}) var<${access}> ${ns}${name}: ${type};\n`;
  }

  const hasCast = needsCast(format, type);
  return (
`@group(${set}) @binding(${binding}) var<${access}> ${ns}${name}Storage: array<${format}>;

fn ${ns}${name}(${args.map((t, i) => `${arg(i)}: ${t}`).join(', ')}) -> ${type} {
  ${hasCast ? 'let v =' : 'return'} ${ns}${name}Storage[${args.length ? arg(0) : '0u'}];
${hasCast ? `  return ${makeSwizzle(format, type, 'v')};\n` : ''
}}
`\.
```

VirtualTable

```
- render: (namespace, ...) ⇒ string

- uniforms?: DataBinding<T>[],
- storages?: DataBinding<T>[],
- textures?: DataBinding<T>[],

- bindingBase?: number,
- volatileBase?: number,
- namespace?: string,
```

Render template

Data bindings

Resolved bindings at **link**-time

With per-stage **visibility**

For a **vertex & fragment** pair

Virtual Module

string = lazy

```
  const access = readWrite ? 'storage, read_write' : 'storage';

  if (args === null) {
    return `@group(${set}) @binding(${binding}) var<${access}> ${ns}${name}: ${type};\n`;
  }

  const hasCast = needsCast(format, type);
  return (
`@group(${set}) @binding(${binding}) var<${access}> ${ns}${name}Storage: array<${format}>;

fn ${ns}${name}(${args.map((t, i) => `${arg(i)}: ${t}`).join(', ')}) -> ${type} {
  ${hasCast ? 'let v =' : 'return'} ${ns}${name}Storage[${args.length ? arg(0) : '0u'}];
${hasCast ? `  return ${makeSwizzle(format, type, 'v')};\n` : ''
}}
`\.
```

VirtualTable

```
- render: (namespace, ...) ⇒ string

- uniforms?: DataBinding<T>[],
- storages?: DataBinding<T>[],
- textures?: DataBinding<T>[],

- bindingBase?: number,
- volatileBase?: number,
- namespace?: string,
```

Render template

Data bindings

Resolved bindings at link-time

...and **PipelineLayout**

With per-stage **visibility**

For a **vertex & fragment** pair

Virtual Module

string = lazy

```
  const access = readWrite ? 'storage, read_write' : 'storage';

  if (args === null) {
    return `@group(${set}) @binding(${binding}) var<${access}> ${ns}${name}: ${type};\n`;
  }

  const hasCast = needsCast(format, type);
  return (
`@group(${set}) @binding(${binding}) var<${access}> ${ns}${name}Storage: array<${format}>;

fn ${ns}${name}(${args.map((t, i) => `${arg(i)}: ${t}`).join(', ')}) -> ${type} {
  ${hasCast ? 'let v =' : 'return'} ${ns}${name}Storage[${args.length ? arg(0) : '0u'}];
${hasCast ? `  return ${makeSwizzle(format, type, 'v')};\n` : ''
}}
`\.
```

VirtualTable

```
- render: (namespace, ...) ⇒ string

- uniforms?: DataBinding<T>[],
- storages?: DataBinding<T>[],
- textures?: DataBinding<T>[],

- bindingBase?: number,
- volatileBase?: number,
- namespace?: string,
```

Render template

Data bindings

Resolved bindings at link-time

...with **minimum binding size**

...and **PipelineLayout**

With per-stage **visibility**

For a **vertex & fragment** pair

Data Binding

Lambda Source

```
- shader: ParsedBundle,

- length: number,
- size: VectorLike,
- version: number,

- bounds?: DataBounds,
- colorSpace?: ColorSpace,
```

Data Binding

Shader lambda
has no dimensions

Lambda Source

```
- shader: ParsedBundle,

- length: number,
- size: VectorLike,
- version: number,

- bounds?: DataBounds,
- colorSpace?: ColorSpace,
```

Data Binding

Shader lambda
has no dimensions

Length & Size
(may be tensor)

Lambda Source

```
- shader: ParsedBundle,

- length: number,
- size: VectorLike,
- version: number,

- bounds?: DataBounds,
- colorSpace?: ColorSpace,
```

Data Binding

Shader lambda
has no dimensions

Length & Size
(may be tensor)

Bounds for positions
ColorSpace for colors

Lambda Source

```
- shader: ParsedBundle,

- length: number,
- size: VectorLike,
- version: number,

- bounds?: DataBounds,
- colorSpace?: ColorSpace,
```

Data Binding

Storage Source

```
- buffer: GPUBuffer,
- byteOffset?: number,
- byteLength?: number,
- version: number,

- format: UniformFormat,
- type?: ParsedBundle,

- length: number,
- size: VectorLike,

- bounds?: DataBounds,
- colorSpace?: ColorSpace,

- readWrite?: boolean,
- volatile?: number,
```

Data Binding

GPUBuffer is not introspectable

Storage Source

```
- buffer: GPUBuffer,
- byteOffset?: number,
- byteLength?: number,
- version: number,

- format: UniformFormat,
- type?: ParsedBundle,

- length: number,
- size: VectorLike,

- bounds?: DataBounds,
- colorSpace?: ColorSpace,

- readWrite?: boolean,
- volatile?: number,
```

Data Binding

GPUBuffer is not introspectable

Format & Size (may be < allocated)

Storage Source

```
- buffer: GPUBuffer,
- byteOffset?: number,
- byteLength?: number,
- version: number,

- format: UniformFormat,
- type?: ParsedBundle,

- length: number,
- size: VectorLike,

- bounds?: DataBounds,
- colorSpace?: ColorSpace,

- readWrite?: boolean,
- volatile?: number,
```

Data Binding

GPUBuffer is not introspectable

Format & Size (may be < allocated)

Bounds for positions
ColorSpace for colors

Storage Source

```
- buffer: GPUBuffer,
- byteOffset?: number,
- byteLength?: number,
- version: number,

- format: UniformFormat,
- type?: ParsedBundle,

- length: number,
- size: VectorLike,

- bounds?: DataBounds,
- colorSpace?: ColorSpace,

- readWrite?: boolean,
- volatile?: number,
```

Data Binding

GPUBuffer is not introspectable

Format & Size (may be < allocated)

Bounds for positions
ColorSpace for colors

Read/write access

Storage Source

```
- buffer: GPUBuffer,
- byteOffset?: number,
- byteLength?: number,
- version: number,

- format: UniformFormat,
- type?: ParsedBundle,

- length: number,
- size: VectorLike,

- bounds?: DataBounds,
- colorSpace?: ColorSpace,

- readWrite?: boolean,
- volatile?: number,
```

Data Binding

Texture Source

```
- texture: GPUTexture,
- view?: GPUTextureView, // or default
- sampler: GPUSampler |
  GPUSamplerDescriptor | null,
- version: number,

- layout: string, // eg. 'texture_2d'
- format: GPUTextureFormat,
- size: VectorLike,

- mips?: number,
- variant?: string, // eg. 'textureSample'
- colorSpace?: ColorSpace,
- absolute?: boolean,
- comparison?: boolean,
- aspect?: GPUTextureAspect,
- volatile?: number,
```

Data Binding

Texture(View) may be paired with **Sampler**

Texture Source

```
- texture: GPUTexture,
- view?: GPUTextureView, // or default
- sampler: GPUSampler |
  GPUSamplerDescriptor | null,
- version: number,

- layout: string, // eg. 'texture_2d'
- format: GPUTextureFormat,
- size: VectorLike,

- mips?: number,
- variant?: string, // eg. 'textureSample'
- colorSpace?: ColorSpace,
- absolute?: boolean,
- comparison?: boolean,
- aspect?: GPUTextureAspect,
- volatile?: number,
```

Data Binding

Texture(View) may be paired with **Sampler**

Type & Size

Texture Source

```
- texture: GPUTexture,
- view?: GPUTextureView, // or default
- sampler: GPUSampler |
  GPUSamplerDescriptor | null,
- version: number,

- layout: string, // eg. 'texture_2d'
- format: GPUTextureFormat,
- size: VectorLike,

- mips?: number,
- variant?: string, // eg. 'textureSample'
- colorSpace?: ColorSpace,
- absolute?: boolean,
- comparison?: boolean,
- aspect?: GPUTextureAspect,
- volatile?: number,
```

Data Binding

Texture(View) may be paired with **Sampler**

Type & Size

MIPs & sampler **variant()**

Texture Source

```
- texture: GPUTexture,
- view?: GPUTextureView, // or default
- sampler: GPUSampler |
  GPUSamplerDescriptor | null,
- version: number,

- layout: string, // eg. 'texture_2d'
- format: GPUTextureFormat,
- size: VectorLike,

- mips?: number,
- variant?: string, // eg. 'textureSample'
- colorSpace?: ColorSpace,
- absolute?: boolean,
- comparison?: boolean,
- aspect?: GPUTextureAspect,
- volatile?: number,
```

Data Binding

Texture(View) may be paired with **Sampler**

Type & Size

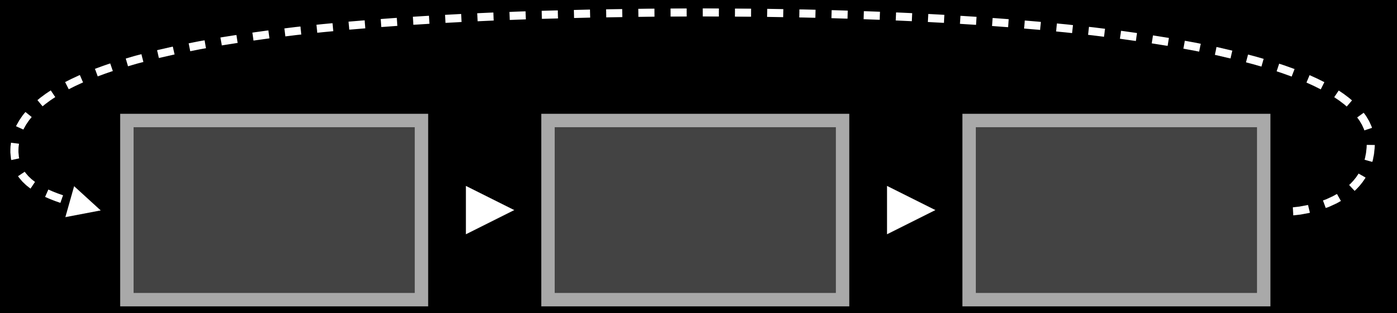
MIPs & sampler **variant()**

Absolute UVs
Depth **comparison**
Stencil aspect

Texture Source

```
- texture: GPUTexture,
- view?: GPUTextureView, // or default
- sampler: GPUSampler |
  GPUSamplerDescriptor | null,
- version: number,

- layout: string, // eg. 'texture_2d'
- format: GPUTextureFormat,
- size: VectorLike,

- mips?: number,
- variant?: string, // eg. 'textureSample'
- colorSpace?: ColorSpace,
- absolute?: boolean,
- comparison?: boolean,
- aspect?: GPUTextureAspect,
- volatile?: number,
```

Target + History



Render / Storage / Texture Target

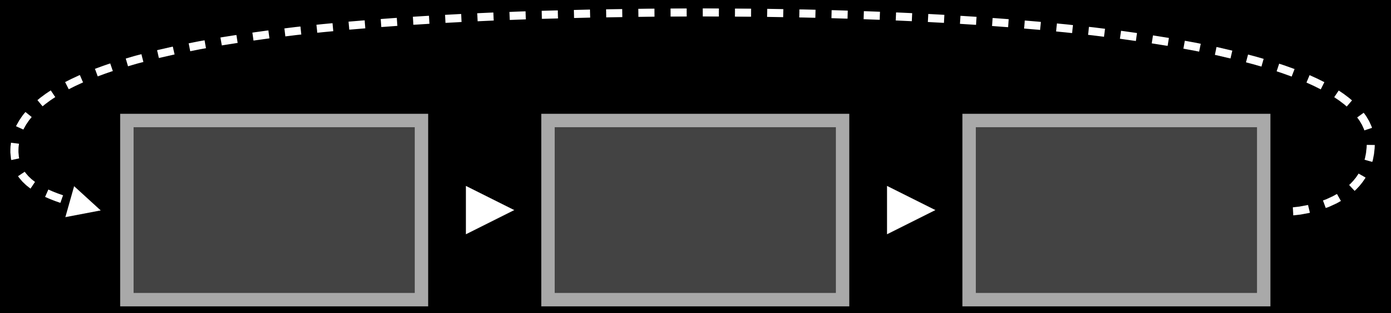
- …

- `history: T[]`
- `swap: () ⇒ void`

- `volatile: N`

Target + History

N **cyclic** buffers
Access is **relative**



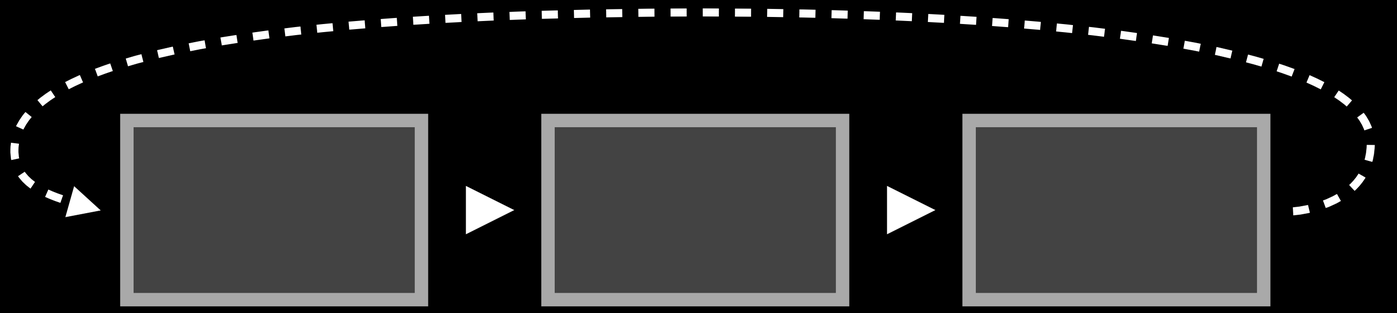
Render / Storage / Texture Target

```
- ...

- history: T[]
- swap: () ⇒ void

- volatile: N
```

Target + History

N **cyclic** buffers
Access is **relative**

Imperative swap
on **dispatch**



Render / Storage / Texture Target

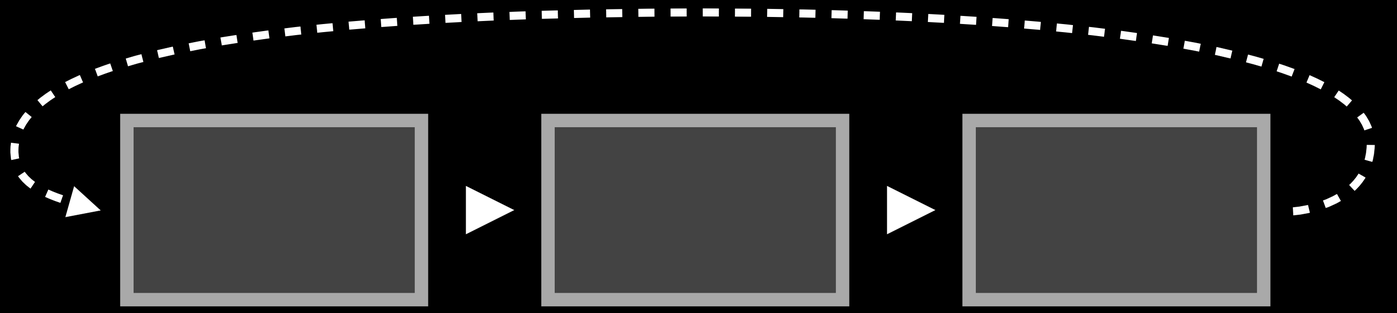
- …
- `history: T[]`
- `swap: () ⇒ void`
- **`volatile: N`**

Target + History

N **cyclic** buffers
Access is **relative**

Imperative swap
on **dispatch**

Volatile bind group cycle
= **`lcm( ... volatiles)`**



Render / Storage / Texture Target

- …
- `history: T[]`
- `swap: () ⇒ void`
- `volatile: N`

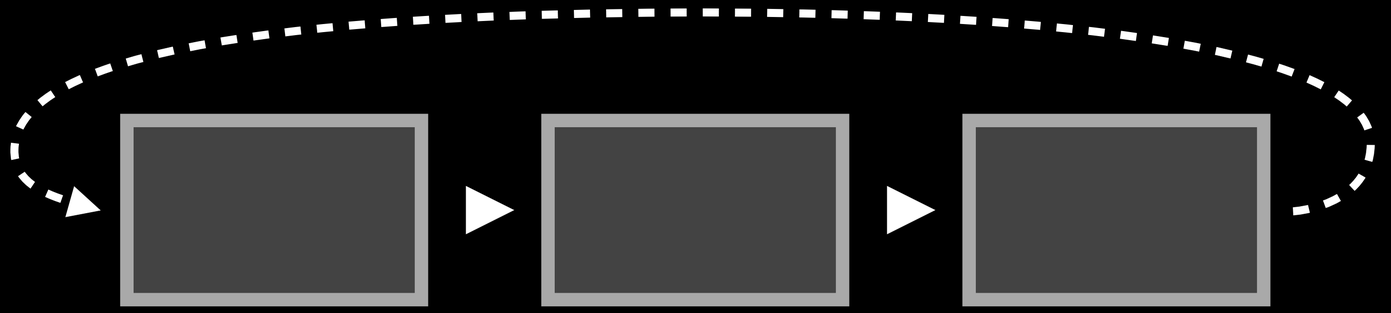
Target + History

N **cyclic** buffers
Access is **relative**

Imperative swap
on **dispatch**

Volatile bind group cycle
= **`lcm( ... volatiles)`**

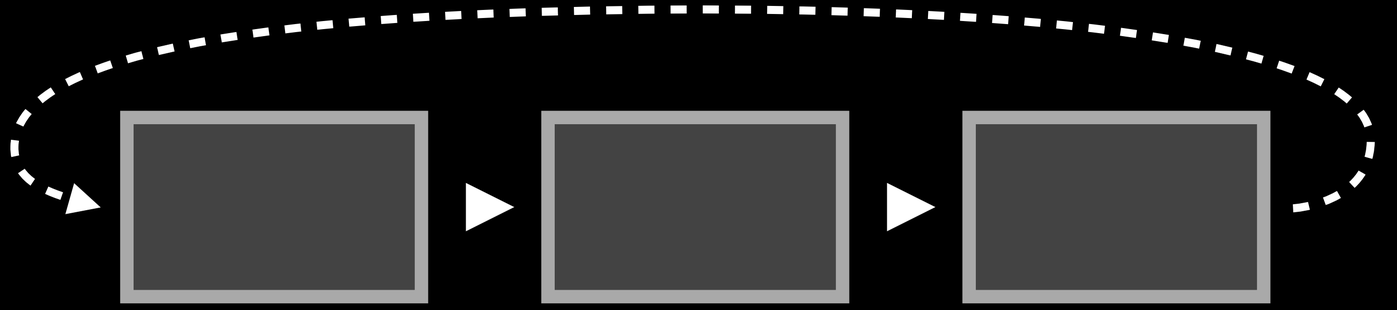
Mini-LRU eviction



Render / Storage / Texture Target

- …
- `history: T[]`
- `swap: () ⇒ void`
- `volatile: N`

Target + History



N **cyclic** buffers
Access is **relative**

Imperative swap
on **dispatch**

Volatile bind group cycle
`= lcm( ... volatiles)`

Mini-LRU eviction

Render / Storage / Texture Target

- …
- `history: T[]`
- `swap: () ⇒ void`
- `volatile: N`

Also **resizable atlas:**
absolute: true
volatile: 1