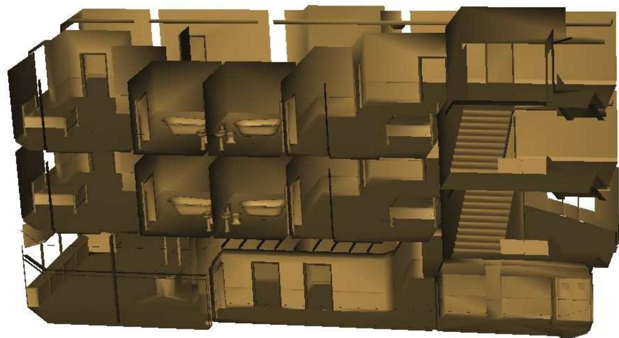
Convex Hull Covering of Polygonal Scenes for Accurate Collision Detection in Games

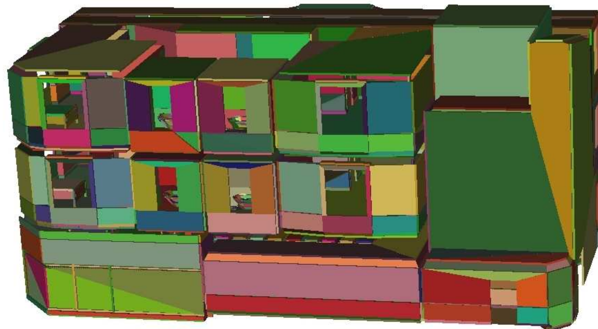
Rong Liu*
Graphics, Usability, and Visualization Lab
School of Computing Science
Simon Fraser University, Canada

Hao Zhang†
Graphics, Usability, and Visualization Lab
School of Computing Science
Simon Fraser University, Canada

James Busby ‡
Radical Entertainment



(a) A building model used in computer games.



(b) Convex hull covering computed by our algorithm.

Figure 1: A result of convex hull covering. (a) A complex building mesh used in games, where front and top walls are culled to reveal the interior structures. The building contains a disconnected collection of closed and open mesh pieces with highly non-uniform tessellations. (b) The convex hulls obtained, shown in different colors, collectively cover the building geometry (they may overlap, hence a *covering*), but do not take away any original game playing space — this is our *accuracy* requirement. The original model has 14,608 polygons and the algorithm returned 3,137 convex hulls. Although the convex hull count is still high due to the strict accuracy requirement, about 80% of collision entity reduction (triangles to convex hulls) still provides great potential to lower the computation cost of collision detection.

Abstract

Decomposing a complex object into simpler pieces, e.g., convex patches or convex polyhedra, is a well-studied geometry problem. A well constructed decomposition can greatly accelerate collision detection since intersections with and between convex objects are fast to compute. In this paper, we look at a particular instance of the convex decomposition problem which arises from real-world game development. Given a collection of polyhedral surfaces (possibly with boundaries, holes, and complex interior structures) that model the scene geometry in a game environment, we wish to find a small set of convex hulls such that colliding objects in the scene against such a set of convex hulls produces the same game behavior as colliding against the original surfaces.

The vague formulation of the problem is due to the difficulty of defining the space accessible by the objects involved in the game play. Under reasonable assumptions, we arrive at a set of conditions for valid convex decomposition and develop a construction algorithm via greedy merging driven by patch compactness. We show that our validity conditions ensure valid collision-related game behavior. The effectiveness of our decomposition algorithm is demonstrated through real examples from game development. To the best of our knowledge, no previous convex hull decomposition or surface decomposition algorithms were designed to handle the type of models we consider or be able to compute a set of convex hulls that ensure accurate collision detection results.

*e-mail: lrong@cs.sfu.ca
†e-mail:haoz@cs.sfu.ca
‡jbusby@radical.ca

1 Introduction

One of the most challenging tasks in computer games is fast and accurate collision detection [6]. A typical game environment is modeled by a collection of triangle meshes representing the characters or other movable objects such as automobiles, and the scene geometry. The latter is assumed to be static, consisting of scene support polygons and static objects, e.g., trees, walls, or other building infrastructures. Collisions can happen between characters and the scene geometry or between characters themselves. Note that we purposely distinguish between the scene support and a terrain, as the former is not necessarily a height field; it is however an open mesh piece, possibly with holes. In real-world scene data, each object can be an open mesh, modeled by its own mesh piece, and the pieces may interpenetrate each other.

In this paper, our problem is to decompose the scene geometry into patches whose convex hulls will facilitate collision computations. First, the number of convex hulls should be small, which would allow collision detection algorithms, e.g., the Gilbert-Johnson-Keerthi (GJK) algorithm [7], to run more efficiently. More importantly, we demand that the use of the resulting convex hulls for collision detection should reproduce the same game behavior as when the original surfaces are used. In short, we say that the obtained convex hulls are *accurate*.

In practice, although the mesh triangle count can be quite high, there are typically large regions that are planar or convex. Combining the triangles in such regions and replacing them by convex hulls

can significantly reduce the computational complexity of identifying colliding object pairs. It is worth noting that in order to maximize efficiency, it is also necessary to build a bounding volume hierarchy (BVH) of the convex hulls and integrate it into the collision engine for fast hierarchical culling. However, this topic is rather standard and not a concern in this paper. We will instead only focus on generating accurate convex hulls.

In addition to efficiency, convex hulls are also advantageous over triangles for collision resolution, since convex hulls, unlike triangles, have a clearly defined interior. Take for example a thin wall standing on a flat landscape. As a triangle mesh, the front and back sides of the vertical wall surfaces will have opposing face normals. If a character traveling at high speed collides with the wall, it is likely to generate contacts with triangles on both sides due to deep penetration, because collision detection is performed only at discrete and relatively large time steps in order to achieve good real-time performance. These contacts will have opposing normals, which makes it impossible to resolve them simultaneously. This causes the character to get stuck in the wall. If we replace the wall triangles with a single convex hull, where the interior of the wall is inside that hull, the contacts generated by collision will have consistent normal directions (pointing to the outside of the convex hull) and the collision resolution routine will successfully correct the penetration, without the character being stuck.

As mentioned above, a key requirement on the generated convex hulls is that the collision detection results derived from them need to be accurate. This means that the convex hulls should faithfully cover the original mesh surfaces. Since the scene support geometry that we are interested in is typically of large scale, even a small approximation error may cause significant visual discrepancy between the collision result and the actual geometry seen by a player. This requirement precludes direct application of previous shape decomposition approaches, e.g., approximate convex decomposition [10] or many methods for surface or part-based mesh segmentation [17]. The accuracy requirement also implies that the convex hulls must not pose any unreasonable obstructions to a character. Unreasonable obstruction is not an issue for closed objects. As long as the convex hulls are contained within an object, nothing will go wrong. In our case, however, we need to cope with meshes that are open or contain holes and interior structures. Convex hulls need to be carefully constructed so that any space accessible by a character during the game must not be culled away by the convex hulls. Consider a cube-like building with a small opening entry. Previous algorithms would likely build a single convex hull for the entire building, closing the entry and solidifying the internal space. A concrete result from our decomposition algorithm can be found in Figure 1. We elaborate on this in Section 3.

Our problem is related to the problem of decomposing a surface into convex patches considered by Chazelle et al. [4], where a surface patch is convex if it lies entirely on the boundary of its convex hull. Finding the solution with the minimum number of patches is NP-hard due to “global failure”. Global failures can occur as a result of possible “twists” in the shapes, e.g., consider a spiral surface, even if all the edges in a patch are locally convex, the patch itself may not be. Although our solution does not aim for the optimal solution, it still has to face potential global failure in order to ensure the accuracy of the convex hulls computed. This is illustrated in Figure 2. We thus resort to a heuristic, based on greedy merging, to solve our mesh decomposition problem. Note also that since our solution will be applied to an actual game engine, issues pertaining to the conceptual simplicity and ease of implementation have been taken into consideration as well.

Our main contributions include a definition of a new convex decomposition problem with respect to *accurate collision detection behavior*, a concrete set of conditions for achieving such accuracy, and an algorithm for constructing the convex hulls. We also show

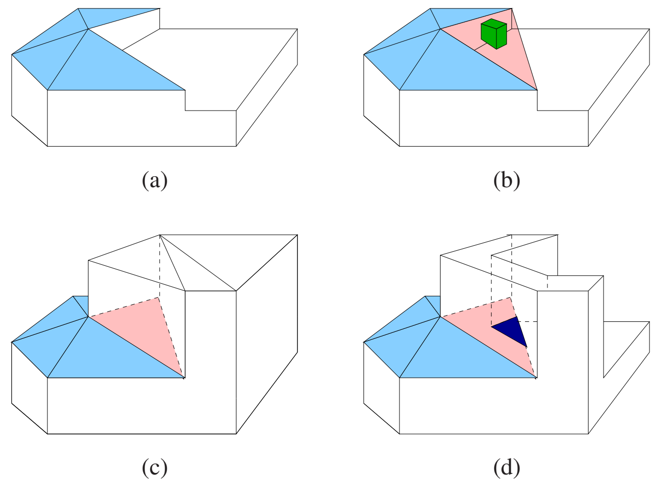


Figure 2: Synthetic examples to illustrate several cases our mesh decomposition algorithm needs to deal with. (a) and (b) show that the convex decomposition of Chazelle et al. [4] does not solve our problem. In (a), the light blue patch is convex since it lies entirely on its convex hull. However, the convex hull contains an extra face, the pink triangle in (b). Using such a convex hull for collision detection will cause an object, e.g., the green block in (b), to “sit in air”, which is an invalid game behavior. The pink triangle would be valid if it lies “inside” the scene support, as shown in (c). However, a “global failure” can occur, as in (d), where the obstruction of the pink triangle to the game environment (indicated by the dark blue region on the pink triangle) cannot be detected based only on geometric information local to the triangle or to the light blue patch.

that a compactness measure helps improve the quality of our results. To the best of our knowledge, no previous convex hull decomposition or surface decomposition algorithms were designed to handle the type of mesh models we attempt to deal with, or could compute a set of convex hulls that ensure accurate collision detection results.

The rest of the paper is organized as follows. After discussing related works in Section 2, we formulate our convex decomposition problem and set up constraints which model the accuracy requirement in Section 3. Section 4 gives detailed coverage on our algorithm and relevant implementation issues. Experimental results are presented in Section 5. Finally, we conclude and discuss possible future works in Section 6.

2 Related work

Our problem falls into the category of mesh decomposition or mesh segmentation. Mesh segmentation can either be of surface-type or part-type [17]. The latter strives to find meaningful components of a geometric model, where the definition of meaningful components is vague and it typically ties in with shape semantics — this problem is different from ours. Surface-type mesh segmentation is more relevant. In works related to mesh parameterization, remeshing, or surface simplification, it is often desirable to identify mesh patches that are geometrically simple, e.g., using criteria such as planarity, convexity [9, 10], compactness [9], constant curvature [14], or low curvature variation [18]. Many proposed techniques, e.g., via region growing, are referenced in the survey by Shamir [17] and we refer readers to that coverage. An important distinction to be made is that the criteria mentioned above are computed based on the patch itself; while a local criterion cannot be found for our problem to guarantee the correctness of the resulting convex hulls. Local criteria are relatively easy to define and more efficient to check. In our work, we need to incorporate global constraints and develop efficient heuristics to check for them.

Previous convex decomposition algorithms can be divided into two categories as in [5]. The first category considers a solid polyhedron T and the goal is to decompose it into multiple solid convex pieces $\{H_i\}$ such that $T = \bigcup_i H_i$ and $H_i \bigcap H_{j,j\neq i} = \emptyset$. This problem has long been studied in computational geometry [1, 3, 16]. A similar case is studied in computer graphics [13, 19] with the relaxed constraint $T \subset \bigcup_i H_i$; namely the solid convex pieces are allowed to overlap, but collectively they contain the polyhedron tightly. Convex decomposition of this type depends on the volume of the polyhedron and does not fit in our application, since the mesh surfaces we need to handle are typically not closed and contain interior structures as well.

The second category of convex decomposition algorithms work on a polyhedral surface S and aims to decompose it into convex patches $\{p_i\}$, with $S = \bigcup_i p_i$ and $p_i \bigcap p_{j,j\neq i} = \emptyset$. A patch p_i is convex if it lies entirely on the boundary of its convex hull. Our work falls into this category. It is proved by Chazelle et al. [4] that finding the minimum set $\{p_i\}$ is NP-hard and they hence resort to several heuristic methods, one of which is later applied in [21]. The closest work to ours is described in [5], in which the authors add the constraint $S \bigcap H_i = p_i$, where H_i is the convex hull of p_i. This additional constraint amounts to requesting that the convex hulls do not contain in its interior any part of the surface S. If S is closed, this condition implies that the resulting convex hulls are entirely contained inside S, guaranteeing correct collision detection. Unfortunately, this does not suffice when handling open surfaces: holes may be closed and surfaces may be extended beyond the boundary by the output convex hulls. Moreover, the special case when p_i is in 2D is not discussed by the authors. If the blue patch in Figure 2(a) were flat, the algorithm in [5] would allow it to be formed and hence lead to unreasonable collision behavior as shown in plot (b). We have to address all these issues in our solution.

Finally, we mention that to accelerate collision detection, it is desirable to make use of a bounding volume hierarchy (BVH). Christer Ericson's book [6] on real-time collision detection has detailed coverage on BVH's and extensive surveys on collision detection in general also exist, e.g., [8, 11, 12]. Building a hierarchy, however, is not the problem we wish to address in this work. We are only interested in building low-level primitives from the triangles which allow for accurate collision detection. A BVH based on these low-level primitives, convex hulls in our case, can be constructed in the same way as in [5] to further speed up collision detection.

3 Convex hull covering

The convex hulls sought in our work are for real-time collision detection in 3D video games. We are mainly interested in computing convex hulls for mesh data which model scene geometry, such as scene support, e.g., terrain, walls and other obstacles, building infrastructures, etc. These objects are often of large-scale and quite complex; see Figure 1, 9, and 10 for a few examples.

Given a triangulated representation $\mathscr{M} = \langle V, E, F \rangle$ of the mesh surfaces, where V, E, F are the vertex, edge, and face (triangle) sets, respectively, we wish to compute a set of convex hulls $\mathscr{C} = \{H_i\}$ out of V, so that the collision results obtained by testing objects against $\mathscr{C}$ are the same as those obtained by testing against $\mathscr{M}$. This is a hard constraint on *accuracy*, which we detail in Section 3.1. At the same time, the efficiency of collision detection should be improved as much as possible; this is discussed in Section 3.2. Section 3.3 outlines the algorithmic challenges we face to motivate our approach. In our application, the input mesh $\mathscr{M}$ is required to be orientable and free of self-intersection; it does not have to be connected. Typically, $\mathscr{M}$ contains a set of connected components, each describing its own surface, open or closed. Our algorithm computes a convex hull covering for each component separately, since it needs connectivity information to merge patches iteratively. However the validity condition checks described in Section 4 have to be carried out on the entire mesh $\mathscr{M}$.

3.1 Accuracy

The accuracy of the produced convex hulls is imposed on collision behavior rather than geometry alone. Specifically, we need $\mathscr{C}$ to be accurate in the sense that when a character interacts with the scene, the collisions triggered using the facets in $\mathscr{C}$ remain consistent with those triggered using the faces in F. Also, since we are dealing with meshes representing large scale scene geometry, unlike approximate convex hull decomposition algorithms, e.g., [10], our tolerance for error is much lower — even a small amount of error on large-scale scene geometry can cause visible unnatural game behavior. To move towards a constructive algorithm, we first interpret accuracy into three requirements:

1. **Reproducibility:** Surfaces of the original mesh should be reproduced by the convex hull facets.

2. **Accessibility:** Convex hulls in $\mathscr{C}$ must not enclose any portion of the original mesh surface in their interior.

3. **Obstruction Free**: Convex hulls in $\mathscr{C}$ should not cause any *unreasonable* obstructions to a character's movement.

Although related, these requirements are not quite the same and are meant to deal with different scenarios. The three of them collectively ensure that the collision behavior will be the same when the convex hulls are used. The first two requirements guarantee that the original scene surfaces are entirely present and available to the characters. However, the way the characters interact with the surfaces may still be affected due to the obstructions caused by the resulting convex hulls, as in Figure 2. Thus the third requirement is introduced and in that we wish to distinguish between reasonable and unreasonable obstructions. Though in theory a character can access any empty region in the embedding space of the scene geometry, game semantics dictate that many regions in the scene are not supposed to be reached, e.g., spaces below a terrain. Obstructions occurring in such regions are acceptable and we call them reasonable. This rather imprecise definition will become clearer in Section 4 when the validity conditions of the convex hulls are described.

Except for these three requirements, output convex hulls are free to take any form or occupy any region in the space. By giving such freedom to convex hull formation, the algorithm is able to produce fewer convex hulls, for better efficiency.

3.2 Efficiency

Collision detection is typically carried out in two phases. At the broad phase, the task is to determine all pairs which can collide with each other. At the narrow phase, the actual collision results, i.e., point of contact, penetration depth, etc., are computed. A smaller number of convex hulls implies less overhead for the broad-phase detection. Therefore, our algorithm should strive to minimize the number of convex hulls, the hull count, in $\mathscr{C}$. On the other hand, the accuracy requirement places several hard constraints on the resulting convex hulls, which can potentially resist reduction on hull count. However, as we stated earlier, these constraints are only concerned with accurate game behavior and not geometric approximation, e.g., a resulting convex hull can cover a large empty space (as long as the space is not supposed to be accessible by a character). Such flexibility serves to effectively reduce the hull count.

3.3 Algorithmic challenges

Convex decomposition algorithms may be volume-based: convex primitives are computed to approximate the volume bounded by mesh surfaces [13, 15, 19]. Meanwhile, it is also practiced to decompose the shape into non-concave parts, where the concavity

measure is based on the distances from mesh vertices to their *corresponding* convex hull facets [9, 10]. Based on these measures, heuristics searching for the global optimal solution can be implemented either top-down, via recursive bisection [20], or bottom-up, e.g., via region growing [9].

Top-down decomposition approaches tend to work well when a global quality measure is available, e.g., mesh volume or concavity. Such measures are only reliable when the meshes are relatively simple and nicely shaped. Unfortunately, the mesh models we need to deal with do not often have clearly defined volumes, neither are the correspondences between mesh vertices and hull facets well defined. They often come with boundaries, holes, tunnels and the surfaces they represent can be rather flat with only local fluctuations or twisted like a helix. Moreover, meshes in games often have complex interior structures. For example, it is not clear how to decompose the building mesh in Figure 1 into valid pieces using a top-down approach in a meaningful manner. For this reason, we have opted for a patch merging approach working bottom-up. Roughly speaking, the algorithm iteratively merges (connected) mesh patches, starting with the original scene triangles, into larger ones $\mathscr{P} = \{p_i\}$. The resulting convex hulls $\{H_i\}$ are the convex hulls of these patches. During the merging process, each pair of adjacent patches need to be evaluated to ensure that the accuracy requirement is maintained; otherwise the two patches will not be merged. Thus instead of considering volume approximation or vertex-to-hull distances, we define *mergeability conditions* specifically to solve the decomposition problem we face.

Mergeability: Recall that the "obstruction free" requirement demands that the convex hulls should not unreasonably obstruct a character's movement. One trivial way to accomplish this is to require each patch in $\mathscr{P}$ to be convex and planar, i.e., it is a 2D convex polygon lying entirely on the original mesh surface. If output convex hulls have to have non-zero volume, such convex polygons can be easily extruded into prisms along the negative local surface normal direction. Although accurate, this simplistic approach can greatly compromise efficiency. In many cases, further merging is possible. Figure 3(a) illustrates such a case. Imagine that the curve shown is a cross-section of a terrain where the arrow indicates the surface normal. The two blue patches p_1 and p_2 are valid to merge, as its convex hull (shown as the light blue trapezoid) sits below the terrain and is an acceptable obstruction. Similarly, the two orange patches are also mergeable. On the other hand, (b) shows an "inverse" case where neither the blue nor the orange patches are mergeable, as the resulting convex hulls would block the passage to the bottom part of the surface. Note that we frequently use 2D profile to illustrate 3D situations in this paper. Be aware that although curves may not be adjacent in a 2D illustration, the patches they represent may be.

(a) P_1 and P_2 are mergeable. (b) Non-mergeable.

Figure 3: Mergeability of patches depends on game semantics.

As we can see, mergeability is something that is related to the game semantics, e.g., what space is accessible and what is not, or in other words, whether a convex hull has the potential to obstruct a character's movement in an unreasonable manner. We shall make these precise next. Note here that it is solely for illustration purposes that we used the term "above" or "below". In practice, such concepts, along with "inside" and "outside" are all ill defined globally due to the complexity presented by our mesh data.

4 ALGORITHM

Our algorithm starts out by classifying each mesh face into a single patch and initializing the merging cost between each adjacent pair. Given any two adjacent patches, denoted by p_1 and p_2, their merging cost is set to infinite if they are not mergeable. Mergeability requires the merged patch to be *valid* so as to produce accurate collision (see Section 4.1). If p_1 and p_2 are mergeable, their merge cost is the inverse of the *compactness* (see Section 4.2) of p, the resulting merged patch. The merging procedure works in a greedy manner: in each iteration, it merges the pair with the smallest cost and update the costs of affected pairs. This process iterates until all the mergeable pairs are exhausted.

4.1 Validity conditions for merging

Let p be the resulting patch by merging p_1 and p_2 and let H be the convex hull of p. We define p to be valid, equivalently, p_1 and p_2 are mergeable, if the following four conditions hold; these conditions give precise geometric interpretations to the three requirements in Section 3.1 for ensuring collision detection accuracy.

1. H contains no mesh vertices or edges in its interior: By the accessibility requirement, H cannot contain any portion of the original mesh surface. Algorithmically, we check whether H contains any mesh vertices or edges. If so, p is invalid. Testing if a mesh vertex is inside a convex hull involves running sidedness tests against hull facets. As for a mesh edge, it suffices to check if it penetrates the convex hull. The only possible case in which a surface portion is enclosed by H but not detectable via the above checks is when the portion is from the interior of a mesh triangle and the three vertices and edges are outside the convex hull. However, this implies self-intersection of the input mesh, which we disallow.

Note that any patch that has both concave and convex edges, which would be non-convex in the sense of Chazelle et al. [4], will be deemed invalid by this condition. However, it does more as it is a global criterion, e.g., it ensures that a generated convex hull is not penetrated by any scene polygon, even one that is far away.

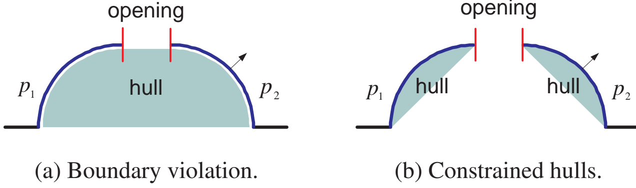
2. Mesh face normals consistent with facet normals in H: While condition 1 ensures that the entire mesh surfaces lie on the facets of the resulting convex hulls, this condition demands consistency in face orientations. The two conditions together guarantee that p is a convex patch, in the sense of Chazelle et al. [4] and consistent with the orientation of the mesh faces. To see that condition 2 is necessary in this context, consider a mesh piece whose Gaussian curvature is negative everywhere.

In fact, both the reproducibility and accessibility requirements are completely handled by these two conditions. In particular, meshes with interior structures or inner surfaces (e.g., a cube whose face normals point inward) are correctly handled. If interior structures are present, the outer convex hull will not be formed as this would violate condition 1. Inner surfaces will be broken into individual pieces contributing to separate convex hulls, so that the interior space is still accessible. What remains now is to model the "obstruction free" requirement.

3. H contains no boundary constraint bars: If the convex hull H extends beyond a surface boundary, either an inner or an outer one, an unreasonable obstruction occurs. Imagine that a character may walk beyond the mesh boundary and stand on some facets of H, causing wrong game behaviors, as illustrated in Figure 4(a). If the convex hull of p_1 and p_2 is formed, it will close the opening. To prevent H from extending itself beyond a boundary, we utilize *boundary constraint bars*.

Each boundary bar is a line segment passing through the center of the corresponding boundary edge and perpendicular to the adjacent mesh face. With these constraint bars guarding along the boundaries, H will always get intersected by some of them if it

extends beyond a boundary, as shown in (a). Accordingly, the algorithm only needs to check whether any boundary bar penetrates or is contained in H. If so, p is invalid and merging is disallowed. Figure 4(b) shows the correctly generated convex hulls, with the opening intact. We make boundary bars perpendicular to their adjacent faces. This is equivalent to bounding the hulls along the face normal direction. We found this to work well for our application. A user may choose to tilt a bar more toward the negative side of the face to constraint the hull more aggressively.



(a) Boundary violation. (b) Constrained hulls.

Figure 4: Use of boundary constraint bars to prevent unreasonable obstruction. (a) Obstruction occurring at an opening by merging p_1 and p_2. The convex hull is invalid as it is intersected by the red boundary constraint bars. (b) Correct result.

Note that so far in our discussion we have assumed that H is a 3D polyhedron. When p is planar, the QHull package [2] we rely on would return H as a convex 2D polygon, in which case the first two conditions are satisfied automatically. As for condition 3, it reduces to testing whether H is intersected, in its planar interior, by any boundary bars.

Unreasonable obstructions may also occur even when the mesh is completely closed, as shown in Figure 5. Assume that H is a polyhedron. Figure 3(b) shows a situation where an obstructing convex hull can completely sit above (locally) the surface. This suggests that the mesh surface and hull facets have inconsistent normals — which should have been caught by condition 2. Since the convex hull cannot extend beyond the boundary (guarded by condition 3), obstructions at the surface interior could only happen when H cuts through the mesh surface. In this case, H has to contain certain mesh vertices or edges, as illustrated by Figure 5, and this is caught by condition 1. Therefore, we only need to take care of cases where H is a 2D convex polygon, which calls for the last condition.

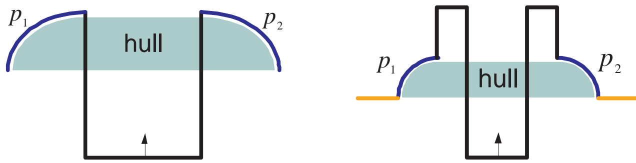
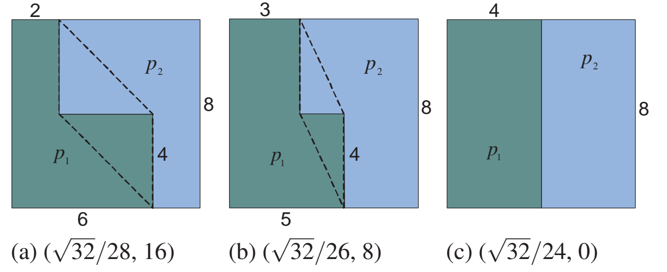


Figure 5: Unreasonable obstructions occur over surface interiors when the relevant convex hull cuts through the mesh surface, in which case condition 1 is violated.

4. No triangles touch the interior and sit on the negative side of H: Consider the two orange patches in Figure 3(b) and the right plot in Figure 5. In neither of these two cases, the two patches should be merged as the resulting convex polygon H causes unreasonable obstruction. In order to detect this type of violation, we check all the mesh faces not belonging to the two patches that also touch the interior of H. If any of these triangles sit on the negative side of H, H is invalid. As in the right plot of Figure 5, H is cut through by some of these triangles. In Figure 3(b), H has such triangles with an edge lying on it but the third vertex sits on the negative side of it. Note that in this condition, the triangles touching H on its boundary are not considered.



(a) $(\sqrt{32}/28, 16)$ (b) $(\sqrt{32}/26, 8)$ (c) $(\sqrt{32}/24, 0)$

Figure 6: Correlation between patch compactness and overlapping. The three plots show three different configurations of two patches with increasing compactness in a square with side length 8. The dashed parallelogram in the first two plots indicate the overlapped regions of the two patches. Under each plot, the numbers in the brackets are the compactness of both patches and the area of the overlapped region, respectively.

If all the above four conditions are satisfied, the accuracy requirement for collision detection using the convex hulls obtained from the merged patches is fulfilled.

In retrospect, the unreasonable obstructions we wish to avoid are mainly caused by planar patches and boundaries. It is worth pointing out, however, that as convex hulls inevitably solidify some empty spaces in the scene, theoretical correctness of collision behavior cannot be guaranteed. In Figure 4(b) for example, the two resulting hulls still take their interior regions away from the scene. However, the reasoning is that a character is not supposed to interact with those two regions; otherwise the mesh should have interior surface at the back side of the two blue patches, in which case the two hulls would not have been formed in the first place. In this particular example, the important factor is to keep the opening. Chances are such an opening in a real game would serve as a "dead hole" or a trigger point to enter another playing scene.

4.2 Compactness-guided merging

With mergeability defined, merging order can simply be random. However, this has its obvious drawbacks. In particular, random merging offers no control over the shape of the patches formed. A valid patch can have a jaggy boundary, leading to more overlapping between the convex hulls. Consequently, if a collision occurs inside an overlapping region on the mesh surface, then it is necessary to pinpoint the exact triangles that cause this collision and extra efforts are needed to check associated triangles with all the overlapped convex hulls. For such efficiency concerns, we wish to reduce overlapping between the resulting convex hulls on the mesh surfaces. This can be accomplished by greedy merging based on a *compactness* measure. The compactness of a patch p is given by:

$$compactness(p) = \frac{\sqrt{A(p)}}{B(p)},$$

where $A(p)$ and $B(p)$ denote the surface area and the boundary length of the patch p, respectively. In 2D, the maximum compactness is achieved by a circular disk. Note that the compactness of p is infinite when p is closed, as $B(p) = 0$. It follows that closed patches are preferred. Figure 6 shows a simple example in 2D. It is easy to see that compact patches tend to have less jaggy boundaries. As a result, convex hulls of these patches overlap less.

One possible alternative to measuring the compactness of a patch is through the compactness of its convex hull defined based on the hull's volume and surface area [9]. However, this measure is not suitable for our application since again the patches being formed during the merging iterations usually do not have a clearly defined enclosed volume. Additionally, it is only the overlapping on the

original mesh surface that can cause concern. For this reason, it is better to measure the compactness of the patch surface itself rather than the compactness of its convex hull.

Another drawback of random merging is its high likelihood to merge in the wrong order, causing poor results. We have found that the compactness score tends to help alleviate this problem. The rationale behind is that in general less compact patches are more likely to cause separation (due to jaggy boundaries) that reduces the overall mergeability among patches. A simple 2D example is shown in Figure 7, while 3D examples can be found in Figure 10.

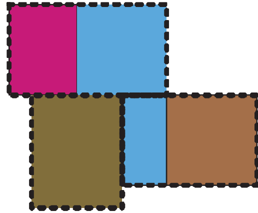


Figure 7: Patch compactness and merging order: In this example, the two cyan patches are merged first, producing a non-compact patch with a jaggy boundary. The jaggy boundary separates the patches which otherwise could be merged. Due to this merge, a total of four patches are generated, while the ideal case should have only three patches as indicated by the dashed rectangles.

4.3 Practical issues

Numerical errors: For a practical implementation, several issues deserve mentioning. The first one is the necessity to handle numerical errors. Our algorithm frequently needs to run sidedness and intersection tests. It would be error prone if these results were simply based on the sign of corresponding distances; this is due to numerical errors or unintentional positional drift of the vertices during design of the mesh models. In our implementation, a tolerance specified by the user is used.

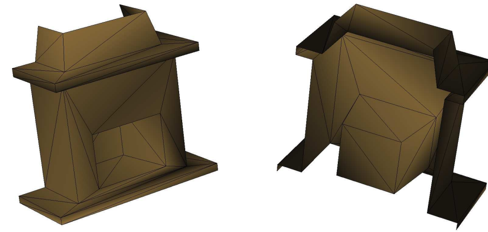
Speeding up validity checks: As validity condition checks must be executed each time two patches are merged, it would be expensive if all the vertices, edges, faces and constraint bars were to be examined. We rely on space partitioning to speed up this process.

Convex hull output: The final patches after the merging process can be either in 3D or planar. When a patch is planar, the convex hull returned is a convex polygon. As the collision engine requires convex hulls with non-zero volume, the final 3D convex hull will be constructed as a prism by duplicating an identical convex polygon translated along the negative surface normal direction and connecting the corresponding vertices.

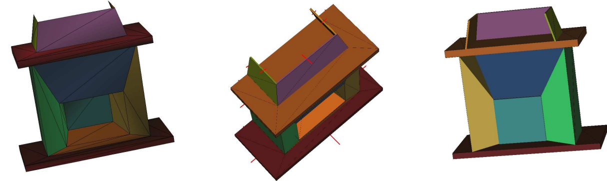
Material properties: So far we have only focused on the geometry of the mesh data and the convex hulls can overlap with each other. In computer games, mesh faces may be associated with non-geometrical properties, e.g., material types. When such face properties come into play, the overlapping on the original mesh surface may cause problems. For example, since each convex hull is supposed to have only one material type, if two convex hulls with different material types overlap on the mesh surface, the collision events triggered inside this overlapping region will not be able to decide the right material type for computing the response. To address this problem, we can simply add a virtual boundary between faces with different materials by adding boundary constraint bars (Section 4.1, condition 3) at their common edges.

5 Results

In this section, we first demonstrate the ideas of our algorithm with simple examples. We also show results obtained on complex mesh data used in real-world games.

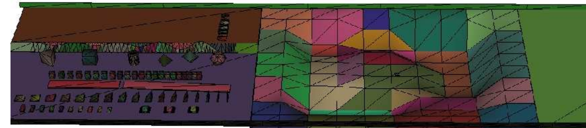


(a) Front view. (b) Back view.

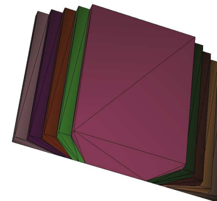


(c) Resulting patches. (d) Hulls (top view). (e) Hulls (back view).

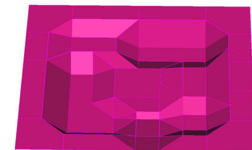
Figure 8: Convex hull covering result on a simple fireplace model.



(a) Overview of decomposed patches on playground.



(b) Patches for a stair.



(c) Hulls for middle region.

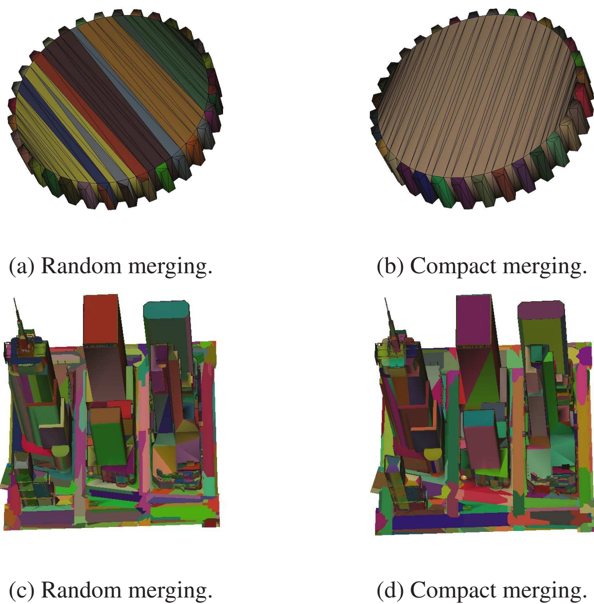
Figure 9: Convex hull covering result on a playground model.

Figure 8 shows our convex hull covering result on a simple fireplace model. The top row gives the front and back views of the model. Since the mesh is single-sided, (b) is generated by reversing the triangle normal for proper lighting. The resulting patches are color coded in (c). As we can see, the algorithm successfully merges the patches with the full capacity and leaves the burning chamber open. Figure (d) shows from atop the convex hulls of each patch. We see that the top three patches are constrained by the boundary bars, without being merged together, and the unreasonable obstruction immediately along the boundary are avoided. We also notice that faces belonging to the top panel form a single patch as its convex hull is "behind" the scene and not reachable by any character, such as Santa Claus.

The algorithm is also tested on a playground and the results are presented in Figure 9. Figure 9(a) shows the overall result. Notice how the wavy part in the upper left region is decomposed into small valid pieces. For the middle region of the playground, we observe that the patches around the four corners are separated nicely to prevent the resulting convex hulls from forming a cover over the

lower surfaces. Figure 9(b) shows a close-up of a stair object in the playground. The meaningful patches are formed despite the non-uniform tessellation. The convex hulls of the middle region of the playground are shown in (c). It can be verified that the accuracy requirement is strictly respected.

In Figure 10, we demonstrate the advantages of compactness-based merging over random merging. The gear shown in the top row has 32 teeth. Random merging has joined some faces from a teeth to those from the central part, as shown in (a). Compactness-based merging successfully decomposes the teeth and the central part into their own patches, generating the ideal result. The second row in Figure 10 shows results from a similar test applied to a block of buildings. Pay attention to the patches on the streets: compact merging effectively produces patches with less jaggy boundaries, hence less overlapping between the resulting convex hulls. Note that the reason that compact merging has still produced some fragmented and jaggy patches in this example is mainly due to the small fluctuations on the street and bad tessellation.

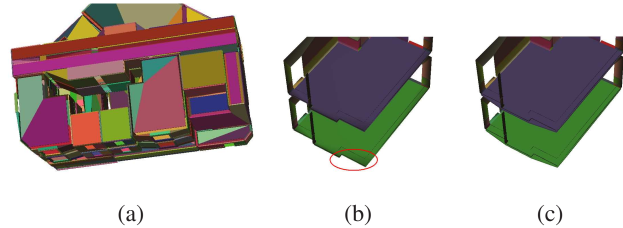


(a) Random merging. (b) Compact merging.

(c) Random merging. (d) Compact merging.

Figure 10: Comparison between random merging and compactness-based merging. The top row shows the comparison for a gear with 32 teeth. The random merging result (a) contains 43 patches, while the compactness-based merging (b) achieves the ideal result with 33 patches. The second row illustrates the patches obtained on a street block, where compactness-based merging also leads to more compact ones, especially those on the streets.

Figure 1 shows a complex building model with interior structures and its convex hull covering generated by our algorithm. Note that the building has only interior walls and for illustration purpose, those back facing triangles are culled. As we can see, the convex hulls generated reproduce the original surface and they also manage to keep the openings and interior space intact. Figure 11 presents a few close-up views of the result. The top view in (a) demonstrates how the patches are formed to ensure their convex hulls are valid. Through the top opening, we can see that the interior space are not filled by the convex hulls. In fact, the entire space inside the building which should be accessible to a character is completely retained when the convex hulls are used. Figures (b) and (c) particularly demonstrate the capability of our algorithm to reduce the number of convex hulls by differentiating between reasonable and unreasonable obstructions. Note how both floors are turned into a single convex hull against the obstructions introduced inside the encircled region. This is valid since the obstruction is reasonable as it is outside the wall (back facing wall faces not shown), where a character is not supposed to access.



(a) (b) (c)

Figure 11: Some close-up views of convex covering of the building model shown in Figure 1.

One limitation of the current algorithm is its obliviousness to small-scale objects in the scene. Figure 12 shows the decomposition result of a tub inside the building (Figure 1). As we impose hard accuracy constraints on all the geometry, the curvedness of the tub surface results in more patches than necessary as its relatively small scale does free up some room for approximation errors as far as collision detection is concerned. However, the challenge lies in how to detect such a small object and allow for a larger error tolerance on it without affecting the correctness of the result as a whole.

Figure 12: Over-segmentation of a small-scale object.

Timing-wise, our algorithm is fairly efficient with the help of the bottom-up approach and local queries against the frequent convex hull computations and validity checks. Depending on the configuration of the model and sizes of patches being formed, our algorithm runs from tens of seconds to about a minute for meshes with up to 21K faces on a 2.5GHz Pentium CPU.

In terms of the asymptotic computational complexity of the algorithm, we use the QHull algorithm to compute convex hulls; this has a cost of $O(n \log n)$. Due to the strict accuracy requirement, we need to compute convex hulls and perform validity checks at each merging step. In the worst case, each validity check is quadratic in complexity, which is likely inevitable since there may be global failures, calling for global intersection tests. Have said that, it is important to note that we take a bottom-up approach. Thus the problem sizes for convex hull construction, along with other necessary operations, are predominantly small. The use of spatial partitioning also greatly speeds up the validity checks. Therefore in practice, our algorithm is quite efficient, as a heuristic for solving a decomposition problem that is likely NP-hard.

One missing component in the current paper is the actual collision efficiency test on the convex hulls in a real game environment. The reason for this is that the game engine currently does not incorporate a bounding volume hierarchy representation for collision objects and the convex hull collision detection routines are poorly optimized. We can only verify the accuracy requirement of the convex hulls at this moment, but we have strong reasons to believe that the performance can be improved significantly once the above two issues are addressed.

6 Conclusion and future work

In this paper, we address a particular convex decomposition problem for real-world game development: decomposing the scene geometry into convex patches so that the convex hulls of these patches serve as a valid alternative to the original mesh for the purpose of real-time collision detection. The challenge comes from the necessity of minimizing the number of convex hulls as well as maintaining exactly the same collision behavior.

The meshes we need to handle are more complex than those considered in previous algorithms. Complications resulting from meshes with boundaries and interior structures can render typical measures used for convex decomposition, e.g., those based on volume or vertex-to-hull distances, ineffective. Our proposed algorithm is guaranteed to produce valid results while allowing for sufficient freedom in convex hull formation. The latter helps reduce convex hull count. The effectiveness of our algorithm is demonstrated through numerous examples.

In the future, we plan to investigate how to effectively minimize the number of convex hull *facets* as well, not just the number of convex hulls. A small number of facets should help reduce the computational cost of collision detection in the narrow phase. The main question is how to factor it into an optimization process. Another interesting yet challenging problem we wish to look into is to extend the algorithm to handle triangle soups. These representations are desired from a 3D artist's point of view as they allow for more freedom in model design. They are difficult to handle from the perspective of geometry processing since a sufficient description of the underlying surfaces in the model can be difficult to infer.

Acknowledgement The authors would like to acknowledge the support from the Accelerate BC Graduate Research Internship Program and Radical Entertainment. We are also deeply grateful to Dr. David Fracchia and the ATG physics team from Radical Entertainment for their generous help and invaluable suggestions. Models used in Figure 8 and Figure 10(a) were taken from the Princeton Shape Benchmark. We thank the anonymous reviewers for their comments as well.

References

[1] C. L. Bajaj and T. K. Dey. Convex decomposition of polyhedra and robustness. *SIAM J. Comput.*, 21(2):339–364, 1992.

[2] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Trans. Math. Softw.*, 22(4):469–483, 1996. http://www.qhull.org.

[3] B. Chazelle. Convex partitions of polyhedra: a lower bound and worst-case optimal algorithm. *SIAM J. Comput.*, 13(3):488–507, 1984.

[4] B. Chazelle, D. P. Dobkin, N. Shouraboura, and A. Tal. Strategies for polyhedral surface decomposition: An experimental study. In *Proc. of Symposium on Computational Geometry*, pages 297–305, 1995.

[5] S. A. Ehmann and M. C. Lin. Accurate and fast proximity queries between polyhedra using convex surface decomposition. In *Proc. of Eurographics' 01*, volume 20, pages 500–510, 2001.

[6] C. Ericson. *Real-Time Collison Detection*. Morgan Kaufmann, 2005.

[7] E. Gilbert, D. Johnson, and S. S. Keerthis. A fast procedure for computing the distance between convex objects in three-dimensional space. *IEEE Journal of Robotics and Automation*, 4(2):193–203, 1988.

[8] P. Jimenez, F. Thomas, and C. Torras. 3d collision detection: a survey. *Computers and Graphics*, 25:269–285, 2001.

[9] V. Kreavoy, D. Julius, and A. Sheffer. Model composition from interchangeable components. In *Proc. of Pacific Graphics*, pages 129–138, 2007.

[10] J.-M. Lien and N. M. Amato. Approximate convex decomposition of polyhedra. In *Proc. of ACM Symposium on Solid and Physical Modeling*, pages 121–131, 2007.

[11] M. Lin and S. Gottschalk. Collision detection between geometric models: a survey. In *Proceedings of IMA, Conference of mathematics of surfaces*, pages 602–608, 1998.

[12] M. Lin and D. Manocha. Collision detection. In J. E. Goodman and J. O'Rourke, editors, *Handbook of discrete and computational geometry*, pages 787–807. Chapman & Hall, 2004.

[13] L. Lu, Y. Choi, W. Wang, and M.-S. Kim. Variational 3d shape segmentation for bounding volume computation. *Computer Graphics Forum*, 26(3):329–338, 2007.

[14] A. Mangan and R. Whitaker. Partitioning 3D surface meshes using watershed segmentation. *IEEE Trans. on Visualization and Computer Graphics*, 5(4):308–321, 1999.

[15] S. Quinlan. Efficient distance computation between non-convex objects. In *Proc. of Robotics and Automation*, pages 3324–3329, 1994.

[16] J. Ruppert and R. Seidel. On the difficulty of triangulating three-dimensional nonconvex polyhedra. *Discrete Comput. Geom.*, 7(3):227–253, 1992.

[17] A. Shamir. Segmentation and shape extraction of 3D boundary meshes. In *Eurographics STAR*, 2006.

[18] O. Sorkine, D. Cohen-Or, R. Goldenthal, and D. Lischinski. Bounded-distortion piecewise mesh parameterization. In *Proc. of IEEE Visualization*, pages 355–362, 2002.

[19] R. Wang, K. Zhou, J. Snyder, X. Liu, H. Bao, Q. Peng, and B. Guo. Variational sphere set approximation for solid objects. *Vis. Comput.*, 22(9):612–621, 2006.

[20] H. Zhang and R. Liu. Mesh segmentation via recursive and visually salient spectral cuts. In *Proc. of Vision, Modeling, and Visualization*, pages 429–436, 2005.

[21] E. Zuckerberger, A. Tal, and S. Shlafman. Polyhedral surface decomposition with applications. *Computer and Graphics*, 26(5):733–743, 2002.