Computers & Graphics 114 (2023) 306–315

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Special Section on CEIG 2023

Real-time rendering and physics of complex dynamic terrains modeled as CSG trees of DEMs carved with spheres

Jesús Alonso [a], Robert Joan-Arinyo [b], Antoni Chica [b,*]

[a] Edifici ETSEIB, Diagonal 647, 8a planta, 08028 Barcelona, Spain
[b] ViRVIG - Research Center for Visualization, Virtual Reality and Graphics Interaction, UPC - Universitat Politècnica de Catalunya, Room 138 - Omega Building, C. Jordi Girona, 1-3, 08034 Barcelona, Spain

ARTICLE INFO

Article history:
Received 20 May 2023
Accepted 14 June 2023
Available online 21 June 2023

Keywords:
Terrain modeling
CSG
Terrain erosion

ABSTRACT

We present a novel proposal for modeling complex dynamic terrains that offers real-time rendering, dynamic updates and physical interaction of entities simultaneously. We can capture any feature from landscapes including tunnels, overhangs and caves, and we can conduct a total destruction of the terrain. Our approach is based on a Constructive Solid Geometry tree, where a set of spheres are subtracted from a base Digital Elevation Model. Erosions on terrain are easily and efficiently carried out with a spherical sculpting tool with pixel-perfect accuracy. Real-time rendering performance is achieved by applying a one-direction CPU–GPU communication strategy and using the standard depth and stencil buffer functionalities provided by any graphics processor.

1. Introduction

Terrains are essential in many computer graphics applications where landscapes play a paramount role. Well-known examples are simulators, games and movies. In games, for example, the terrain tends to be the dominant visual element of the scene. In general, terrains are composed of a huge amount of data that entail expensive rendering and interacting processes.

Terrain modeling and rendering have been widely studied in the literature, aiming at achieving both real-time performance and accuracy. There is a large number of publications concerning, for example, procedural methods for synthetic terrain generation, real-time rendering of terrains, realistic rendering of terrains by adding features or eroding terrains. Most previously proposed methods to model and edit terrains with tunnels, caves and overhangs only account for scenarios where the terrain does not change over time. The most relevant merits of these works will be detailed in Section 2.

In this work, we report on an algorithm for rendering landscapes generated by sculpting Digital Elevation Models (DEM) using a sphere as a sculpting tool. The DEM maintains an unmodified version of the terrain surface while spheres dynamically applied to the terrain model carvings and erosion. The landscape is actually modeled as a Constructive Solid Geometry (CSG) tree where the DEM is the deepest node, and the spheres are subtractive nodes. The model easily captures tunnels, overhangs and terrain erosion. The rendering algorithm works on landscapes with an arbitrary number of spheres, does not require replacing spheres with polyhedral approximations, and avoids evaluating the landscape surface. We show how the described approach performs by applying it to synthetic and real terrain examples.

The paper is organized as follows. We first in Section 2 review the work related to the problem we are dealing with that has been described in the literature. Then in Section 3, we define the geometric model used in our approach. Next, we present our approach in Section 4, the validity of the model in Section 5, and the physics system in Section 6. Experimental results are given in Section 7. Finally, we offer a short discussion in Section 8.

2. Previous work

Noticeable progress has been made toward developing techniques to model terrains with caves and overhangs [1]. However, when it comes to destruction, erosion or sculpting techniques that require real-time updating and great accuracy for both rendering and physics collisions, the number of published works is rather scarce. In what follows we overview those works which are closely related to the work described here. We group them according to one of the following topics: (i) DEM and triangulated irregular networks (TIN) hybrid models, (ii) layered and voxel models, (iii) L-systems and patterns, (iv) geology applications, (v) games, (vi) game engine tools and (vii) CSG.

Early works proposed the use of hybrid models by mixing DEM and TIN. For example, the work in [2] applies non-linear deformations to an initial heightfield surface to create procedural

* Corresponding author.
E-mail addresses: jalonso@cs.upc.edu (J. Alonso), achica@cs.upc.edu (A. Chica).

https://doi.org/10.1016/j.cag.2023.06.019

landscapes with overhangs. The work described in [3] describes a data structure that captures the result of applying geotectonic events to a given DEM as a set of heightmaps blended with TIN. In contrast, games simulate caves or tunnels by combining hollow terrains with mesh blended caverns. The main drawback of these approaches is that iteratively modifying a 3D irregular triangulation results in unstable meshes. Thus they apply to terrains that are basically static.

The techniques described in [4,5] model tunnels and terrain overhangs by applying layered terrains while the work in [6] is based on a voxel model. Realistic static terrains can be obtained with these techniques and some optimizations can be done by efficiently storing the data, see for example [7]. However, since both real-time updates and accuracy highly depends on the terrain resolution, working with voxel or pseudo-voxel models are serious drawbacks.

L-systems, grammars and patterns have also been applied to model caves. The work in [8] applies 3D models to generate pattern-based caves using predefined pattern images. Caves are modeled as a combination of two simple independent heightmaps. A base heightmap is generated by the grammar and a reflected copy is used to represent the top part of the cave. Then the final model is a partial overlap of the two heightmaps. A method that applies L-systems and cellular automata to schematic input maps is described in [9]. The output is an irregular 3D mesh. In [10] caves are generated by combining L-systems, metaball carving of a voxel data model and isosurface extraction. Due to the fact that these methods are globally defined, making local changes is rather difficult. Besides, patterns can be clearly identified in the renderings.

Literature devoted to capturing changes in geological structures focuses on the simulation of terrain formations, tectonics, deformation mechanics, and erosion generated by natural phenomena. Thus they do not aim at rendering realistic and dynamically changing 3D terrains at interactive frame rates. The recent work in [11] describes a new method aiming at modeling geological dynamic behavior rendered in real-time.

Games are computer graphics applications where interaction is crucial. In general, changes in the scenario are the result of the user–game interaction, thus a high frame rate when rendering the resulting scene is a must, and computations to update the scene must be reduced as much as possible. Computation reduction is achieved in different ways. Games described in [12,13] allow to destroy objects, vehicles, trees, structures or buildings in the scene. The terrain model is immutable, and changes on it are simulated, for example, by texturing [14–16] or displacement mapping [17].

Recent games involving open-world scenarios like [18,19] combine DEM and 3D meshes. Heightmaps model the basic terrain while caves and terrain irregularities are modeled with 3D meshes.

Several different techniques can be found in the published games to modify the terrain model. Among them, we find games that simulate destruction using additional TIN, games that modify original DEM and voxel-based models. TIN meshes do not allow friendly interaction because they are not built over a regular structure; thus, solving physics and visibility require expensive computing resources. Therefore TIN is rarely used in games. However, the game described in [20] models erosion using TIN.

Games described in [21–23] modify terrains by updating the height values of a DEM. Clearly, these approaches do not allow for creating overhangs and tunnels. Trails, footprints or tire tracks can also be easily associated with DEM using shaders. See for example the games in [24–26].

Voxel-based models have been used in a variety of games to represent destructible terrain [27]. Still, being memory intensive limits them to low-resolution or relatively small scenes. Considering all platform games, voxel-based dynamic terrains and big landscapes allow interaction only for either low resolutions [28,29] or the number of voxels to be updated is small [30]. The games [31,32] allow full environment destruction by combining a low-resolution basic terrain and 3D meshes for terrain irregularities and features that vanish when the terrain is eroded.

Considering game engines, we find tools and plugins that mix DEM with meshes as for example [33]. Other hybrid solutions use DEM and voxel-based models as [34]. The solutions [35–37] offer full environment-destruction possibilities with voxel models plus GPU-accelerated techniques to smooth and enrich the visual appearance.

Classical rendering of terrains modeled as CSG trees follows basically two different strategies. One strategy focuses on extracting a mesh that approximates boundaries [38,39]. These methods usually apply only to static scenes and are unsuitable for interactive editing. Even though there are techniques to efficiently extract meshes from volumetric data [40], it still is too expensive for large scenes, especially if editing is a requirement. Thus this research area is out of the scope of our work. Image-based techniques [41–43] are a widely used technique. This approach is based on applying multiple-pass techniques and requires two or more buffers.

Work in [44] describes an approach that renders boolean combinations of free-form triangulated shapes. The method first transforms the CGS tree into a data structure called *blist* that avoids the need for converting the CSG model into a disjunctive normal form. Then, primitives in the CSG tree are iteratively rendered until all pixels in the image are successfully classified. Two GPU buffers are used. One stores the pixel color according to the current iteration. The second buffer stores the color of the final image.

Recent works focus on taking profit from graphic processors. They achieve interactive visualization by implementing a GPU ray-tracing technique. See for example the work in [45,46] or [47] built upon spatial hashing techniques. These approaches severely depend on the scene's complexity, the spatial scene decomposition and the amount of available local shader memory. Hence, they are not well suited for scenes that change over time.

Table 1 summarizes the abilities of some solid representation methods in terms of our main goals. Specifically, we evaluate the ability to model cavities and tunnels, processing and storage requirements, scalability, dynamic updates performance, in/out test, real-time rendering and accuracy.

Table 1
Qualitative assessment of some solid modeling representations abilities to deal with editing and rendering eroded terrains.

	Implicit solids	CSG	DEM	Hybrid DEM+TIN	TIN	Voxels	Our approach
Cavities	+	+	–	+	++	+	++
Processing	–	a	++	+	–	+	+
Storage	++	+	+	+	a	–	+
Scalability	+	+	+	–	+	a	+
Updating	–	+	+	–	+	+	+
In/Out Test	++	+	++	+	–	++	++
Rendering	–	–	++	+	–	+	+
Accuracy	–	+	a	+	++	–	+

(–) Poor, (a) Average, (+) Good, (++), Very good.

3. The geometric model

We first define the set of points in $\mathbb{R}^3$ to be modeled. Let a DEM be defined within the domain $D = [xmin, xmax] \times [zmin, zmax]$

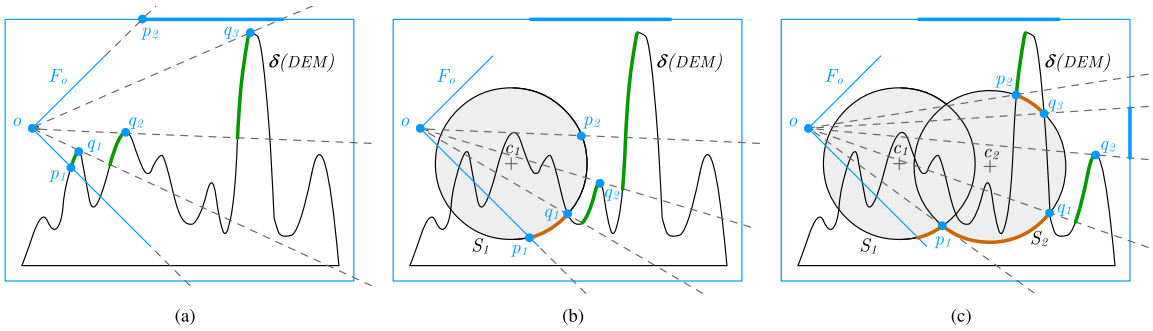


Fig. 1. Algorithm outline. Between points p_1 and p_2, q_i are the points where visibility status can change.

with an associated height function $h_{DEM} : D \rightarrow \mathbb{R}$. The set of points to be modeled is

$$T = \{(x, y, z) | (x, z) \in D \text{ and } Y_{min} \leq y \leq h_{DEM}(x, z)\}$$

where Y_{min} is an arbitrary value equal or less than $min(h_{DEM}(x, z))$. Notice that this set is bounded, closed, and the boundary, denoted from now on as $\delta(DEM)$, does not self-intersect and is trivially orientable.

In these conditions, given a terrain as a DEM, we aim at modeling and rendering terrains with overhangs, tunnels and surface erosions. These modeling operations can easily be performed by carving the DEM with some carving tool. To capture this idea, we model an eroded terrain as a CSG tree where the geometric primitives are the DEM and a sphere as a carving tool. We first consider spheres of fixed radius. In Section 5 we detail the changes needed to deal with spheres of variable radius. The Boolean operator is the difference. The DEM is placed at the tree deepest. Spheres are placed bottom-up according to increasing distances to the viewpoint. Spheres at the same distance are randomly placed within their distance slot.

4. The algorithm

We first give an outline of the algorithm to render the geometric model. Then we explain how we identify the set of surfels that must be considered. To end, we describe the rendering process.

4.1. Algorithm outline

Our approach uses rasterization to compute the set of surfels [48] to be rendered. Surfels are classified using the graphics hardware following the approach described in [44]. Surfel status is stored as a mask in the stencil buffer in the pixel where the surfel will be projected. No sampling of surfels is ever stored.

Consider the 2D DEM embedded in a blue skybox depicted in Fig. 1a where the viewpoint is denoted as o, the current field of view is F_o and dotted lines are rays that emanate from o and go through points where visibility changes. Clearly, the visibility status of surfels can only change at DEM silhouette points within the field of view, say q_i, when the terrain is seen from o. Visible surfels are shown in bold green points on $\delta(DEM)$.

Assume that we place a carving sphere, say S_1, such that the intersection $S_1 \cap$DEM is not empty. See Fig. 1b. As a result of the carving, visibility status can change at either terrain silhouette points or at points where S_1 and the DEM intersect within F_o. These points are p_1 and q_1 in Fig. 1b thus, since the set of points $S_1 \cap$DEM is removed, DEM points on the shortest arc of S_1 that connects p_1 and q_1, shown in brown, are now visible. The final set of surfels the visibility status of which must be updated, is figured out once the set of carving spheres has been considered by traversing the geometric model tree. Fig. 1c illustrates the results after applying two spheres.

4.2. Front and back spheres

In general, only a part of a carving sphere surface is active when performing a carving on the convex surface of a DEM. We generically consider two different parts in a carving sphere: the *front sphere*, S_f, and the *back sphere*, S_b, defined as follows.

As before, o denotes the viewpoint and F_o the field of view. Let p be a point on the carving sphere S with local normal $\vec{n}_p$ and $\vec{r}_p$ the ray from o through p. Fig. 2a illustrates these concepts. The *back sphere* is defined as the subset of points on S that, by carving, are candidates to change the visibility status of DEM surfels, that is

$$S_b = \{p \mid p \in S \text{ and } \vec{r}_p \cdot \vec{n}_p > 0\} \cap F_o$$

In Fig. 2a, S_b is the arc in bold of S. Similarly, the *front sphere* is defined as the subset of points on S that will never change the visibility status of any DEM surfel

$$S_f = \{p \mid p \in S \text{ and } \vec{r}_p \cdot \vec{n}_p < 0\} \cap F_o$$

Surfels candidates to be updated belong to either the $\delta(DEM)$ or to the new terrain surface generated by the carving of S_b. We identify them by applying the crossing parity number [49,50].

Notice that we only need to consider rays within the intersection of the field of view F_o with a cone tangent to S_b and apex o, depicted in Fig. 2a in yellow. A tighter set of rays will be identified in Section 4.4.

Let p be a point on S_b and $\vec{r}_p$ the ray through p. First, we calculate the set of potential surfels generated by $\vec{r}_p$, that is

$$Q_p = \{q | q \in \vec{r}_p \cap \delta(DEM) \text{ and } d(o, q) \geq d(o, p)\}$$

Then we sort Q according to increasing distances from the viewpoint o. Thus p is the first point in Q. Consider the point p on S_b as depicted in Fig. 2b. Counting the parity crossing number on ray $\vec{r}_p$ classifies p as laying inside $\delta(DEM)$. Thus S_b is carving the terrain at point p, and the surfel corresponds to new carved surface.

Now consider the point $q \in Q_q$ and the ray $\vec{r}_q$. See Fig. 2b. Parity crossing number on ray $\vec{r}_q$ classifies q as laying outside $\delta(DEM)$. In these conditions, the surfel to be rendered corresponds to the second point in Q_q, that is, a point on $\delta(DEM)$. Fig. 2c shows the set of surfels where the status visibility changes. Surfels on $\delta(DEM)$ that are no longer occluded by the terrain are shown in green. New surface surfels generated by carving are shown in brown.

4.3. The stencil buffer

The stencil buffer plays an essential role in our approach, it is used to store information concerning the status of some calculations. Three different masks are stored in the stencil buffer.

The first mask, M_{wa}, stores the surfels under study that will be modified. The second mask, M_{count}, stores for each surfel the

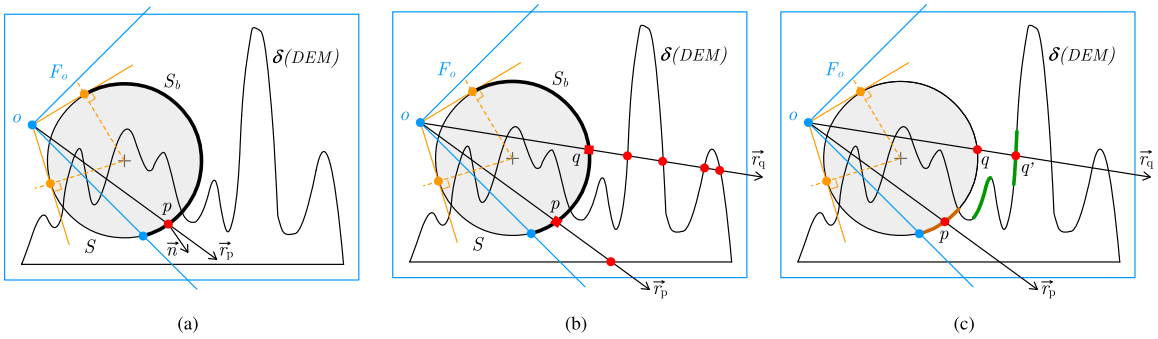


Fig. 2. Only points on the back sphere can generate changes in the visibility status.

parity crossing number as a counter modulo 2 for the number of times its corresponding ray crosses $\delta(DEM)$. The third mask is a binary flag that signals whether some ray through the surfel associated with the pixel in the stencil buffer has already been traced.

4.4. The working area

We call the set of surfels whose visibility status will change as a result of carving the terrain with a single sphere the *working area*. All the pixels from this area and only these pixels will be updated once rendering evaluation is carried out.

Figuring out the working area depends on how the viewpoint o is placed concerning both the carving sphere S under consideration and the terrain surface $\delta(DEM)$. We only consider three possible $S/\delta(DEM)$ relative placements of o: out/out, in/out, in/in. Notice that the case out/in where o is outside S and inside $\delta(DEM)$ would mean that the viewpoint o is completely surrounded by terrain, and nothing could be seen from o even if S carved some bite of terrain. Fig. 3 illustrates the three cases considered for a 2D terrain. The working area is depicted as an arc labeled wa. All the calculations described in what follows are performed in the GPU. The working area is stored in the M_{wa} mask. To compute M_{wa}, we disable the color buffer and set the depth buffer as read-only. Initially, all the mask M_{wa} is set to inactive.

Case out/out. We consider first the case where the viewpoint o is outside of both S and $\delta(DEM)$, as illustrated in Fig. 3a for a 2D terrain. Let R denote the set of rays from o such that for all $\vec{r} \in R$, $\vec{r} \cap S_b \neq \emptyset$ and $\vec{r} \cap \delta(DEM) \neq \emptyset$. Then the working area is the set of points on S_b defined as

$$wa = \{p \in S_b \mid p = \vec{r} \cap S_b,\ \vec{r} \in R\}$$

To reduce as much as possible the need to change the GPU status, the GPU shall always work on backface culling mode. Consequently, we keep two instances of the carving sphere. In one instance, denoted S^+, normals point toward the unbounded space. In the other sphere instance, denoted S^-, normals point toward the bounded space. Then the working area is computed with the next two steps: (1) Rasterize S^+ and for each surfel $s \in S^+$ set $M_{wa}(s)$ to active. (2) Rasterize S^- and for each surfel $s \in S^-$, if $M_{wa}(s)$ is set to active, then set $M_{wa}(s)$ to inactive.

Case in/out. Now we consider the case where the viewer is inside the sphere S and outside $\delta(DEM)$, as shown in Fig. 3b. Notice that in these conditions, there is no need to rasterize the sphere S^+. Thus, we rasterize S^- and, for each surfel s, $M_{wa}(s)$ is set to active.

Case in/in. When the viewpoint o is inside of both the carving sphere under consideration S and $\delta(DEM)$, we need to distinguish two different scenarios. If the intersection of S with any previously considered sphere is empty, the working area is trivially the back sphere S_b. See Fig. 3c.

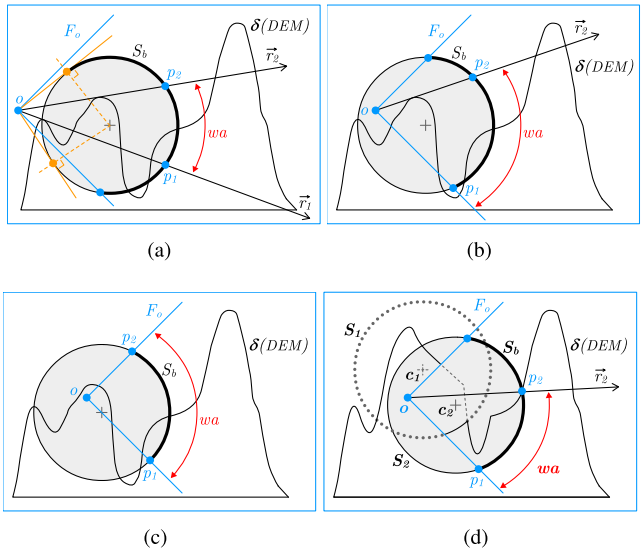


Fig. 3. The working area cases denoted by the arc labeled wa.

Consider now the situation depicted in Fig. 3d where S_1 and S_2 are carving spheres. S_2 is the sphere under consideration, while S_1 is an already considered one. The viewpoint o is inside of both spheres as well as inside $\delta(DEM)$. However, o is outside the carved terrain; therefore, the visibility must be figured out by applying the case in/out.

4.5. Rendering

Once the working area has been identified and stored in the stencil buffer, the next step is to render the eroded terrain effectively. The set of potential surfels Q_p, defined in Section 4.1, is computed by intersecting the ray r_p with $\delta(DEM)$. We use a strategy in which triangles that model $\delta(DEM)$ are sorted according to increasing distances from the point of view, as described in [51]. This technique guarantees that points in Q_p will be also sorted according to the same criterion.

The rendering algorithm to subtract a sphere has three major steps. Assume that the results are stored in the buffer C_sZ_s, which contains both the z values in Z_s and the color values in C_s.

In the first step, Z_s is initialized, assuming that all the surfels in the working area belong to the new terrain carved by the sphere. We rasterize S^- in the Z_s with the z-function set to *always* as long as $M_{wa}(s)$ is active.

In the second step, two actions are carried out. The algorithm calculates the parity crossing number and stores the result in $M_{count}(s)$. Simultaneously, we update C_s and Z_s with the first

visible surface of $\delta(DEM)$ through r_s. To perform this procedure, we set the z-function to the *greater* value, we rasterize $\delta(DEM)$, and we set up the stencil buffer in such a way that computations are done in the pixels matching $M_{wa}(s)$, counting is performed in $M_{count}(s)$, and we only update C_s and Z_s once.

To fix those surfels which were wrongly updated in the previous step, in a third step, S^- is rasterized again. C_s and Z_s are updated according to the parity crossing number stored in M_{count} with the surfels of the back sphere.

Algorithm 1 summarizes the procedure to render our model.

Algorithm 1 Rendering model evaluation

```
1:  function RENDER(model, viewer)
2:      T ← Sort(model.dem(), viewer)
3:      S ← Sort(model.spheres(), viewer)
4:      initialise CsZs
5:      CsZs ← Rasterize(T, z-func=LESS)
6:      for each s ∈ S do
7:          initialise M_wa, M_count
8:          M_wa ← WorkingArea(s, model, viewer)
                        ▷ 1st step: assume complete carving in Zs
9:          Zs ← Rasterize(s_b, z-func=ALWAYS, M_wa)
                        ▷ 2nd step: counting and partial result
10:         setup stencil to count parity and to update CsZs once
11:         CsZs, M_count ← Rasterize(T, z-func=GREATER, M_wa)
                        ▷ 3rd step: update if parity test is even
12:         CsZs ← Rasterize(s_b, z-func=ALWAYS, M_count)
13:     end for
14: end function
```

Fig. 4 shows the concepts introduced in this section using the initial DEM depicted in image (a) and two eroding spheres as illustrated in image (b). In the image (c), the pixels in red represent the front spheres. In the image (d), the pixels in red depict the back spheres. The adjacent set of pixels in yellow and blue color represents the working area of each sphere. The yellow pixels correspond to the parity crossing test with an odd result, while the blue ones to an even result. The last image (e) shows the result of the evaluation rendering. Where we have blue pixels, we place the surfels from the back spheres with a brown texture. In the yellow pixels, we place the surfels beyond the back spheres with respect to the viewpoint. These surfels are no longer self-occluded by the DEM because of the terrain carving. Specifically, on the left sphere, new surfels from the terrain and the skybox are now visible while, on the right one, we can see a new part of the terrain.

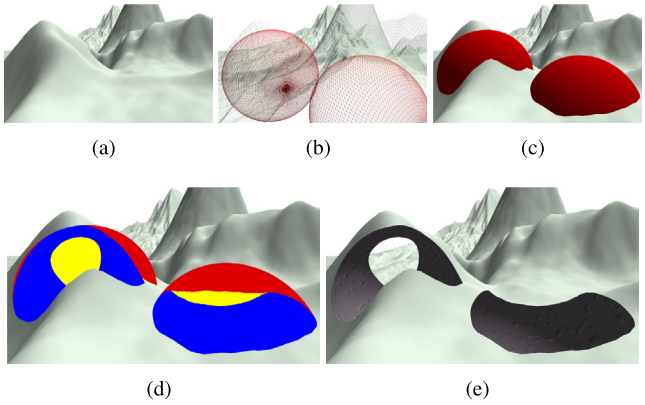


Fig. 4. The rendering evaluation of a DEM with two eroding spheres.

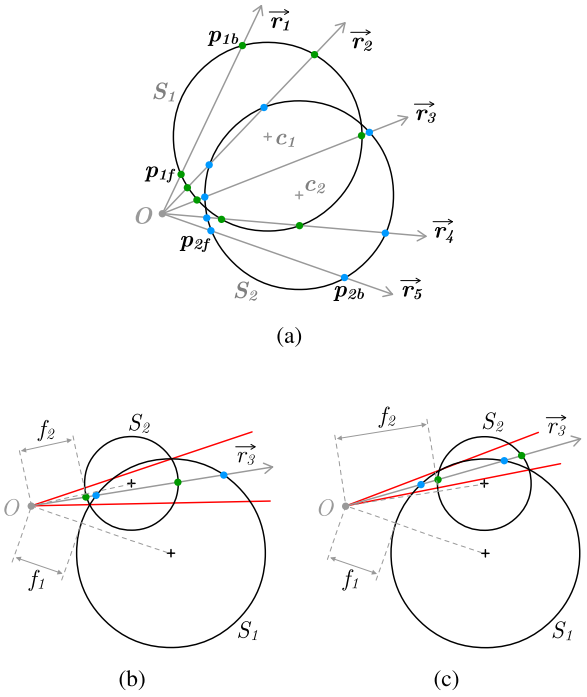


Fig. 5. Interactions between a line of vision with origin O and two spheres S_1 and S_2.

5. Validity of the model

The key points of our approach are the individual and sorted application of the carving spheres and their contribution to the final erosion. As we mentioned, our approach is a CSG model composed of the initial terrain and the set of eroding spheres as primitives. We begin by rendering the initial terrain. Then, we sort the set of spheres and apply them using the subtraction operation.

As a sorting criteria, we use the sorting factor f computed as follows. Given a viewer O with viewing direction $\vec{v}$ and a sphere with center C and radius r, if $\vec{OC} \cdot \vec{v} > 0$ then $f = |\vec{OC}| - r$, $f = -|\vec{OC}| - r$ otherwise. Notice f is a continuous function.

Concerning the erosion of a sphere, we must state that one sphere only deletes material within its volume, and its visible result is limited by its working area. The partial contribution result from a sphere will be incorrect if and only if a later sphere erodes them. The sorted application of the spheres guarantees properly overwriting these possible wrong partial solutions. That is, when a sphere is applied, the erosion is correctly computed but can be eroded with farther erosions.

Next Fig. 5a depicts all possible interactions from a ray of view with origin O with two overlapping spheres S_1 and S_2 with center c_1 and c_2, same radius and sorting factors f_1 and f_2, respectively, such that $f_1 < f_2$, denoted as $\vec{r}_i$, $1 \leq i \leq 5$. The green points refer to the intersections points with S_1, and the blue points to S_2. For each ray and each sphere, the first intersection points belong to the front sphere denoted as p_{1f} and p_{2f}, and the second intersection points belong to the back sphere denoted as p_{1b} and p_{2b}.

Thus, the five possible lists of intersection points from each ray, sorted regarding O are $[p_{1f}, p_{1b}]$, $[p_{1f}, p_{2f}, p_{2b}, p_{1b}]$, $[p_{1f}, p_{2f}, p_{1b}, p_{2b}]$, $[p_{2f}, p_{1f}, p_{1b}, p_{2b}]$, and $[p_{2f}, p_{2b}]$. The erosion intervals for cases $\vec{r}_1$ and $\vec{r}_2$ are $[p_{1f}, p_{1b}]$ and for cases $\vec{r}_4$ and $\vec{r}_5$ are $[p_{2f}, p_{2b}]$. Therefore in these four situations, the result can

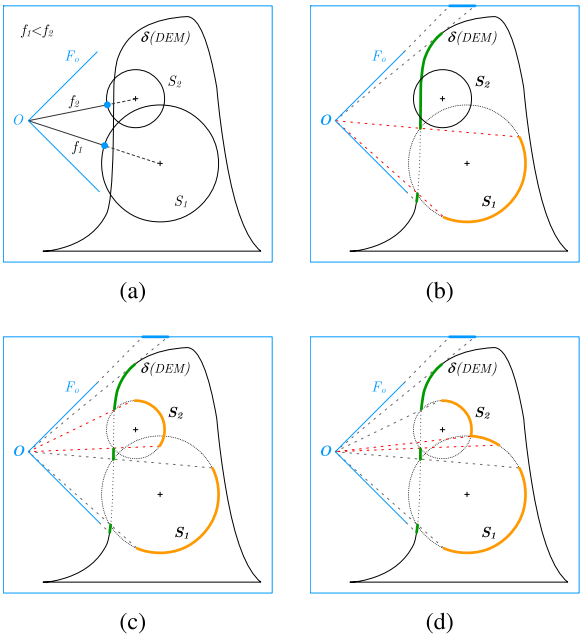


Fig. 6. Appearance and correction of the sorting artifact.

be properly computed regardless of the order we apply S_1 and S_2 due to the fact erosion intervals can be defined by only one sphere. In the case $\vec{r}_3$ though, sorting spheres is fundamental, we need to start with S_1 and finish with S_2. The sorting factor introduced solves this problem. However, when we deal with spheres of different radii, the sorting factor is not enough. See for example Fig. 5b and c. In both cases $f_1 < f_2$ but they have different erosion intervals, $[p_{2f}, p_{1b}]$ and $[p_{1f}, p_{2b}]$, respectively.

A new special treatment is needed to adapt the model to deal with spheres of variable radius. From now on, the area in which the pixels are incorrect due to this issue will be referenced as the *sorting artifact*. First, we define the set of conditions in which it is possible to generate a sorting artifact. Then, we introduce the procedure to address and fix the possible artifacts.

Given a list of sorted spheres with their radius values $S = [S_1(r_1), S_2(r_2), \ldots, S_n(r_n)]$ and the current sphere to be processed $S_i(r_i) \in S$, $1 < i \leq n$, we say a sorting artifact can be produced after applying S_i if $\exists j$, $1 \leq j < i$ such $S_i(r_i) \cap S_j(r_j) \neq 0$ and $r_i < r_j$.

For convenience, we consider lists of sorted spheres of just two intersecting elements. On the one hand, it is not necessary to study spheres which are not overlapping, they do not introduce interferences. On the other hand, the way we process spheres, increasingly and individually, allows us to focus the study by considering the current sphere and one previous random sphere. Thus, the list of sorting spheres can be simplified to $S = [S_j, S_i]$.

Next Fig. 6 depicts a step to step evaluation of the case of the image in the previous Fig. 5b. The sorted list of spheres is $S = [S_1, S_2]$ due to the fact $f_1 < f_2$. Image (b) shows the result after applying S_1 and image (c) after applying S_2. The red dashed lines limit the working areas for each sphere. After these two steps, we can confirm the existence of a sorting artifact. S_1 is not eroding as much as it could due to the fact the algorithm started with S_1, and there are some rays with origin O where $\|\vec{Op_{2f}}\| < \|\vec{Op_{1f}}\|$. At this point, we can check that after applying S_2, there was a previous and bigger sphere intersecting with S_2, that is, S_1. Thus, the sorting artifact effectively could appear and actually appears.

To fix the sorting artifact, once a sphere $S_i(r_i)$ is processed, we repeat the rendering evaluation of all those spheres $S_j(r_j)$ that can create a sorting artifact. That is all spheres S_j such $S_i(r_i) \cap S_j(r_j) \neq 0$ and $r_i < r_j$. Since the sorting artifact is a phenomenon produced due to the spatial relationship between two spheres, no matter the order in which we re-evaluate these S_j spheres. Fig. 6d depicts the final result once this extra step is carried out.

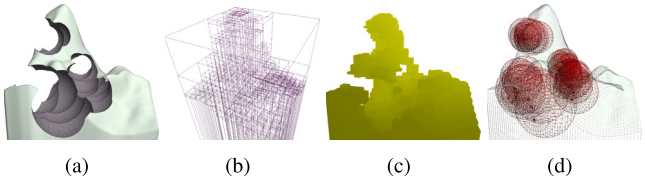


Fig. 7. Representation of the data structures of the broad, mid and narrow phases used to update the physical system model.

6. The physics system

Our physical model uses a specific data structure, allowing us to provide a real-time testing model with broad, mid and narrow phases. The broad phase uses a quadtree spatial data structure of AABBs of the initial terrain, that is, the terrain without any modification. The mid phase maintains a coarse evaluation of the model. It uses a grid where each element is a run-length encoding representation of the solid materials within a column XZ of the terrain. The narrow phase comprises the triangles of the initial terrain and a graph of spheres where each sphere maintains the references to those spheres which intersect. Fig. 7 shows for the specific eroded terrain in the image (a), the broad (b), mid (c) and narrow (d) data structures used for each phase.

In our approach, we capture the continuous motion of entities modeled as spheres. Their continuous collisions can be processed by using as a swept model, the ray defined by two different positions p_1 and p_2 of the center of a sphere. We can easily extend this procedure to a set of points to increase the accuracy of the collision. In what follows, we will use T as the digital terrain model and S as the set of the eroding spheres. Fig. 8 depicts two examples where the swept model defined by segment $\overline{p_1p_2}$ is crossing T and several spheres.

In the physics evaluation of our model, the resulting collision point q can belong to a sphere of S or the surface of T. Both situations are depicted in Fig. 8a and Fig. 8b, respectively. The list of candidate points LC to be the collision point q is defined as follows. Let LPT be the sorted list of points resulting from the intersection between T and $\overline{p_1p_2}$. Let LPS be the sorted list of points resulting from the intersection between the set of spheres of S and $\overline{p_1p_2}$. Let LP be the list of points resulting from the union of LPT and LPS. Points on LP must be kept sorted regarding p_1. See for example in Fig. 8a we have $LP = [q_1, q_2, \ldots, q_7]$. A point $q_i \in LP$ is included in LC if and only if it accomplishes one of these two statements: (1) If $q_i \in LPT$: if p_1 is *out* T we consider q_i if its located in an odd position on LPT, otherwise, if it is located in an even position. (2) If $q_i \in LPS$: let be c the center position of the sphere $S_i \in S$ such as $q_i \in S_i$. We consider q_i if $\overline{q_ic} \cdot \overline{p_1p_2} < 0$.

The final collision point q_i, is the first point from LC such as either (1) $q_i \in LPT$ or (2) $q_i \in LPS$, and is the farthest point connected from spheres of S through $\overline{p_1p_2}$ within T. That is, if $S_i \in S$ and $q_i \in S_i$, q_i is *in* T and q_{i+1} is *out* S_i. See for example in Fig. 8a, $LC = [q_2, q_4, q_5, q_7]$ and the collision point is q_5. In Fig. 8b, $LC = [q_2, q_4, q_6, q_7]$ and the collision point is q_7.

Let $\vec{w}$ be the initial direction $\vec{w} = \overrightarrow{p_1p_2}$. Once we obtain the intersection point q and the unit vector $\vec{n}$ of the collision plane, the new position is updated with q and the new direction $\vec{v}$ with the reflection vector of $\vec{w}$ as $\vec{v} = \vec{w} - 2(\vec{w} \cdot \vec{n})\vec{n}$. By placing the object on position q, we use one frame to simulate damping.

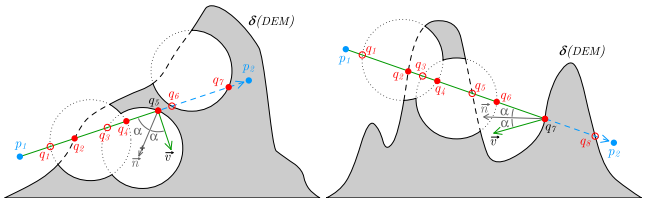


Fig. 8. Collision point given by a sphere (a) and by terrain (b).

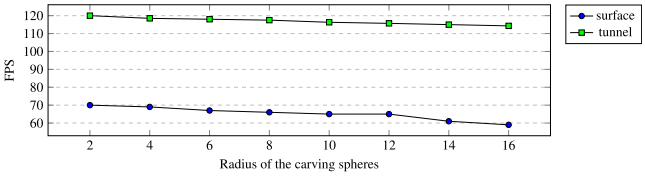


Fig. 9. Influence of the carving sphere radius on the performance.

7. Experimental results

To assess the performance of the model and approach proposed, we have conducted a set of different tests always measuring the performance as the number of frames per second rendered. The tests conducted include: the influence of the carving sphere radius, the number of carving spheres applied, carving holes and overhangs on the terrain, carving tunnels, the terrain resolution, the variable radius performance impact, and dynamic erosions and physics performance of moving actors. We have used a DEM that models Mount Ruapehu and Mount Ngauruhoe, placed in a volcanic zone in New Zealand.

The experiments have been conducted on a PC AMD Ryzen 7 3700X, with 32 GB RAM, featuring an Nvidia Geforce RTX 2070 graphics board with 8 GB. Programs have been developed in Visual Studio under Windows 10. The graphics API used was OpenGL, and the freeglut library was used for events and window management. For all tests a 1024 × 1024 heightfield of 16 bits is used unless otherwise specified. This elevation model is transformed into a terrain of 1024 units in both the X and Z axis, and 256 units in the Y axis. This video [52] captures real-time footage of our tests.

7.1. Radius of the carving spheres

To assess the effect of the carving sphere radius on the performance, we applied two series of 8 tests. Each test included 1000 carving spheres with a constant radius ranging from 2 to 16 units measured on the terrain scale.

The first series only carved on the terrain surface $\delta(DEM)$ while the second series carved tunnels. To consider the worst scenario, the view was always zenithal. By doing so, all spheres must be processed. Performances are collected in Fig. 9. Results for the case of surface carving are labeled *surface* and results for tunnel carving are labeled *tunnel*. The performance when spheres carve tunnels is practically doubled. This is due to the big difference in the number of inner spheres between both scenarios and the fact that from inner spheres, there is no chance to devise a new visible part of the terrain. Thus, for these carving spheres, one step of the algorithm can be avoided. Notice that, in any case, the role played by the radius of the carving sphere in the performance is almost negligible.

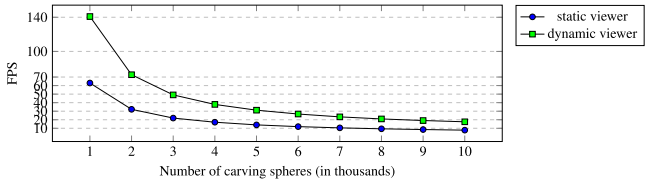


Fig. 10. Rendering performance as a function of the number of carving spheres.

7.2. Number of carving spheres

To study how the number of spheres used to carve holes and overhangs on $\delta(DEM)$ affect the performance, we throw a top–bottom shower of randomly generated spheres that carved the terrain surface $\delta(DEM)$. The test consisted of throwing a series of sets of carving spheres with a number of spheres ranging from one thousand up to ten thousand. The radius of the spheres was always 8 terrain units.

We considered two different scenarios. In the first one, the viewer was placed at a fixed zenithal point. In the second, the viewer moved along an arbitrary path that never collided with $\delta(DEM)$. The performance for the static viewer is shown in the plot labeled *static viewer* of Fig. 10. The number of fps for two thousand carving spheres is 32.2, clearly enough for real-time interaction.

In the second scenario, we made the viewer smoothly move along a path over the terrain. The path allowed the viewer to observe the $\delta(DEM)$ by moving it from one terrain corner to the opposite one while rotating to look in different directions. The plot labeled *dynamic viewer* in Fig. 10 collects the measured number of fps. Here the approach renders 72.7 fps when the number of carving spheres is two thousand.

The fact that when the viewpoint moves the performance is higher than when it is at a fixed place can be attributed to two facts. On the one hand, since the DEM is convex, when the viewpoint is placed at a fixed zenithal point, all the carvings and the whole $\delta(DEM)$ are visible. So the computing load is maximal. On the other hand, when the viewpoint moves along a path, many carvings and parts of the $\delta(DEM)$ are eliminated by frustum culling. Besides, our approach computes from scratch the whole scene on each frame and does not depend on the viewpoint placement.

7.3. DEM resolution

The performance of our approach with respect to the DEM resolution has been assessed applying the test described in Section 7.2 but the DEM is now discretized on a 2048 × 2048 regular grid. We only report performances for the worst case corresponding to the static viewpoint.

Fig. 11 depicts the rendering performance for the two terrain resolutions, 1024 × 1024 and 2048 × 2048. Notice that the plots are almost the same. When the number of carving spheres is two thousand we get 30 fps. When the number of carving spheres is three thousand, the number of fps is still above 20. We conclude that the DEM resolution does not have an impact on the performance of the computations.

7.4. Variable radius performance impact

We tested the impact of variable radius. Specifically, the radii of the spheres used are defined in the interval [3, 14] in terrain units. Table 2 shows the results obtained with a number of spheres ranging from five hundred to three thousand spheres. We

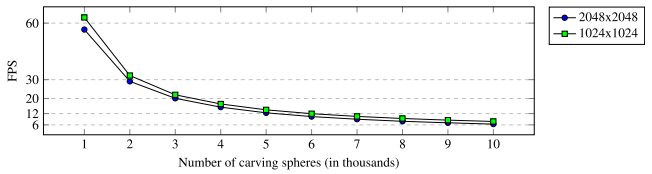


Fig. 11. Impact of the DEM resolution on the approach performance for a static viewpoint.

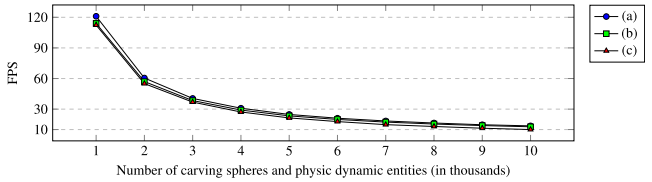


Fig. 12. Physics performance regarding the number of dynamic carving spheres and dynamic entities.

Table 2
Variable radius impact on performance.

Spheres	FPS incorrect	FPS corrected	FPS diff	Spheres re-evaluated
500	129.0	116.0	10.1%	11.5%
1000	64.0	50.0	21.9%	23.2%
1500	44.7	32.6	27.1%	31.8%
2000	33.9	23.0	32.2%	41.4%
2500	27.6	17.7	35.9%	50.5%
3000	23.3	14.0	39.9%	59.7%

only report performances for the worst case corresponding to the static viewpoint.

As we saw in Section 4, to include spheres of variable radius involves an extra step in which reprocessing some carving spheres can be needed. The column labeled as *FPS incorrect* refers to the rendering without this step whereas *FPS corrected* refers to the complete procedure. This extra step impacts the performance of the rendering evaluation and the increasing cost is proportional to the number of spheres intersecting with different radii. As it can be seen in column *Spheres re-evaluated*, the more spheres we add, the more spheres must be processed due to the fact that the chances and number of intersections increase. Notice that if no intersections are present, there is no performance impact.

7.5. Dynamic erosions with physical dynamic entities

The next test we have carried out introduces dynamic carving spheres and dynamic entities on the terrain simultaneously. Fig. 12 shows the resulting performance obtained. Notice that all values on X-axis refer to 50% of carving spheres and 50% of dynamic entities. Specifically, it measures the performance cost of three curves: the initial cost of terrain rendering with carving spheres (a), the performance when we incorporate the entities rendering (b), and the total cost when we also conduct the entities' physics computation (c). We can observe that rendering (a) takes most of the computing time. The contribution of the physical computations in the final frame rate increases linearly from 1% when there are one thousand spheres to 20% when we have ten thousand spheres.

7.6. Physical behavior

To end, we have manually designed several scenarios to check out the correct physical behavior of thousands of bouncings entities. Next, we present two of them, the *TestCave* and the *TestPaths*.

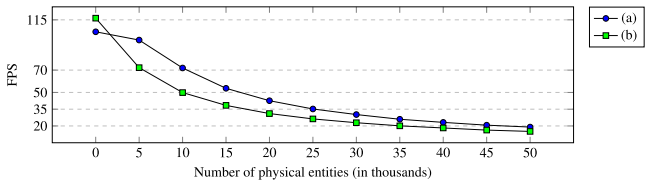
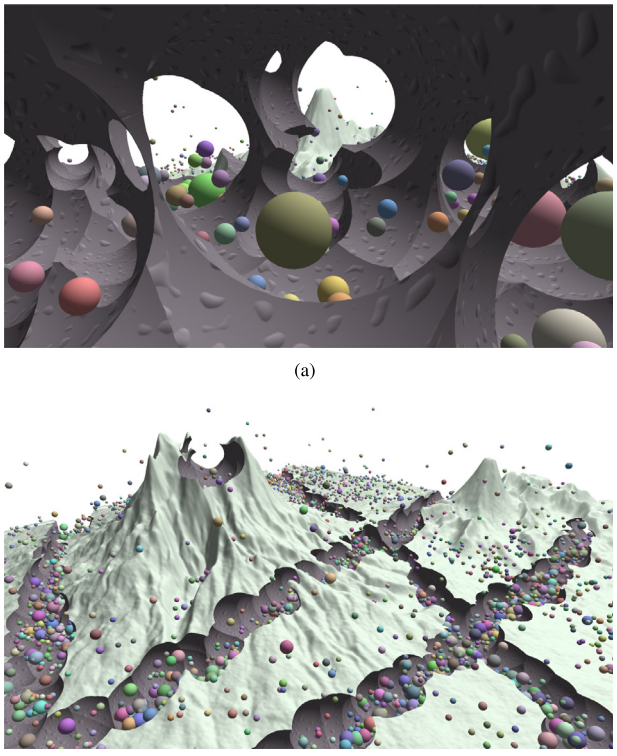


Fig. 13. Performance of the *TestCave* (a) and *TestPaths* (b) experiments.



(a)

(b)

Fig. 14. The *TestCave* (a) and *TestPaths* (b) experiments.

The first experiment consists of a cave with tunnels and holes created with 150 carving spheres. In the second experiment, we used 400 carving spheres to drill different paths. We perform an exhaustive test using up to 50 thousand physical entities. In both experiments, the dynamic entities impact the eroded surfaces as expected. In the experiment *TestCave*, we are above 30 fps with 30 thousand entities and in the experiment *TestPaths* with 20 thousand entities. Using 50 thousand moving objects, we obtain 19.0 fps and 15.1 fps, respectively. Fig. 13 depicts the performance of *TestCave* and *TestPaths*. Fig. 14 shows both experiments.

7.7. General modeling

There is nothing essential in the fact that $\delta(DEM)$ captures a terrain's surface. The approach can be applied to sculpt any coherent closed triangulation free of self-intersections. Fig. 15 illustrates the results output when the approach is applied to sculpt a teapot from a raw stock. Image (a) shows a cubic raw stock modeled as a $\delta(DEM)$. Images (b) and (c) show the result of removing part of the stock. Finally, images (d), (e) and (f) depict the clean fully sculpted teapot. The graphics user interface we have developed is experimental and, consequently, offers a small set of basic sculpting interactions. Hence the teapot has been sculpted with the help of a simple script that places the carving spheres in the 3D space. Specifically, the test consists of a cubic

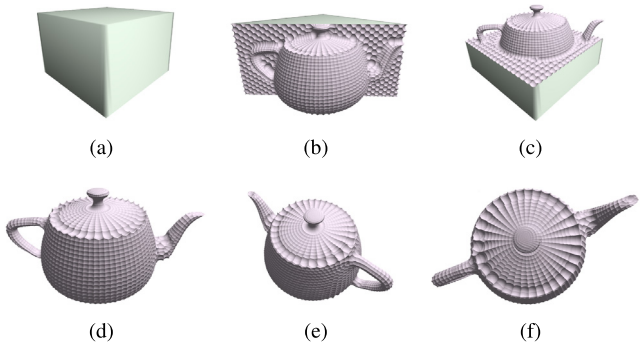


Fig. 15. Teapot sculpted by carving a raw stock with spheres.

DEM of 32 × 32 resolution with 12,539 spheres of fixed radius performing 7.12 fps. Clearly, lesser spheres with variable radii could obtain a similar figure featuring a better performance with an improved user interface.

8. Conclusions

The approach described, where information always flows from the CPU to the GPU, introduces a way to evaluate CSG expressions in rendering different from those we have found in the literature. As a matter of fact, the CSG expression is never explicitly evaluated. Once the viewpoint is fixed, the main idea is, for each carving sphere, to identify the minimum set of surfels in the 3D model, the visibility status of which can change. Partial results are stored in the buffers of the commodity graphics card available on our computer.

No level of detail or any similar acceleration technique has been applied to the $\delta(DEM)$ or the spheres. However, in the worse scenario where carving takes place on the $\delta(DEM)$, the viewpoint is zenithal and the whole terrain is visible, about 30fps are rendered for a number of carving spheres of up to two thousand.

Similar results could be achieved using other techniques at the cost of a performance hit. Trying to perform the difference operation on the meshes of the heightfield and the carved spheres would have to deal with the robustness problems inherent to Boolean operations. Cherchi et al. [53] propose a robust and interactive approach, achieving 25 fps with scenes of 25K total triangles. This is an improvement over current implementations of [54], like the one in libigl [55]. Another option would be to use raytracing to obtain the intervals of each ray intersected by the heightfield and then subtract the carving spheres to get the first intersection. To render the heightfield, we can either raytrace the corresponding mesh, which is costly, or apply a relief map-based raymarcher which will lead to artifacts in the rendering process. Tevs et al. [56] propose a hybrid approach that accelerates the intersection test of a raymarcher while also providing an exact bilinear patch intersection. They report 33 fps for a 1024 × 1024 heightfield. We expect that adding thousands of carving spheres would significantly impact its performance. Even though our approach is limited to sphere carving, it achieves real-time on a scene of 67.823.616 total triangles derived from a 1024 × 1024 heightmap and two thousand carving spheres, providing better performance than these alternatives.

Besides the real-time rendering and editing performance obtained in our proposal, we have successfully incorporated the ability to evaluate the model for physics simultaneously. Tens of thousands of entities can interact with the model with an increasing cost of up to 20% of the rendering cost.

In future work, we would want to improve the performance by identifying clusters of spheres that may be rendered on the same step. The idea is to replace each sphere node in the CSG tree with a cluster of spheres. Then, when considering a cluster, all the possible sequences of spheres in it are rendered so we can severely speed up the performance. Preliminary results are promising, but conceptual aspects concerning how to identify minimal clusters need to be further elaborated.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request

Acknowledgments

This work has been partially funded by Ministeri de Ciència i Innovació (MICIN), Agencia Estatal de Investigación (AEI) and the Fons Europeu de Desenvolupament Regional (FEDER) (project PID2021-122136OB-C21 funded by MCIN/AEI/10.13039/50110001 1033/FEDER, UE).

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.cag.2023.06.019.

References

[1] Galin E, Guérin E, Peytavie A, Cordonnier G, Cani M-P, Benes B, et al. A review of digital terrain modeling. In: Computer graphics forum, vol. 38, no. 2. Wiley Online Library; 2019, p. 553–77.
[2] Gamito M, Musgrave F. Procedural landscapes with overhangs. In: Proc. 10th Portuguese computer graphics meeting. 2001, p. 33–42.
[3] Alonso J, Joan-Arinyo R. The grounded heightmap tree - a new data structure for terrain representation. In: Proceedings of the third international conference on computer graphics theory and applications - volume 1: GRAPP. SciTePress, INSTICC; 2008, p. 80–5.
[4] Benes B, Forsbach R. Layered data representation for visual simulation of terrain erosion. In: Proceedings of the 17th spring conference on computer graphics. Washington, DC, USA: IEEE Computer Society; 2001, p. 80–6.
[5] Peytavie A, Galin E, Grosjean J, Merillou S. Arches: a framework for modeling complex terrains. Comput Graph Forum 2009.
[6] Santamaría-Ibirika A, Cantero X, Salazar M, Devesa J, Santos I, Huerta S, et al. Procedural approach to volumetric terrain generation. Vis Comput 2014;30(9):997–1007.
[7] Cui J, Chow Y, Zhang M. A voxel-based octree construction approach for procedural cave generation. Int. J. Comput. Sci. Netw. Secur. 2011;11.
[8] Boggus M, Crawfis R. Procedural creation of 3D solution cave models. In: Proceedings of the IASTED international conference on modelling and simulation. 2009.
[9] Antoniuk I, Rokita P. Generation of complex underground systems for application in computer games with schematic maps and L-systems. In: Computer vision and graphics. Springer International Publishing; 2016, p. 3–16.
[10] Mark B, Berechet T, Mahlmann T, Togelius J. Procedural generation of 3D caves for games on the GPU. In: Foundations of digital games. FDG, 2015.
[11] Cordonnier G, Cani M, Benes B, Braun J, Galin E. Sculpting mountains: Interactive terrain modeling based on subsurface geology. IEEE Trans Vis Comput Graphics 2018;24(5):1756–69.
[12] Epic Games, Inc. Fortnite: Battle royale. 2017, Game [PlayStation 4, Xbox One, Windows, Macintosh] Epic Games, Inc.
[13] EA DICE. Battlefield V. 2018, Game [PlayStation 4, Xbox One, Windows] Electronic Arts.
[14] Lengyel E. Applying decals to arbitrary surfaces. Game Program. Gems 2001;2:497–509.
[15] Kaneko T, Takahei T, Inami M, Kawakami N, Yanagida Y, Maeda T, et al. Detailed shape representation with parallax mapping. In: Proceedings of ICAT, vol. 2001. 2001, p. 205–8.

[16] Policarpo F, Oliveira MM, Comba JL. Real-time relief mapping on arbitrary polygonal surfaces. In: Proceedings of the 2005 symposium on interactive 3D graphics and games. 2005, p. 155–62.
[17] Szirmay-Kalos L, Umenhoffer T. Displacement mapping on the GPU-state of the art. In: Computer graphics forum, vol. 27, no. 6. Wiley Online Library; 2008, p. 1567–92.
[18] BioWare Montréal. Mass effect: Andromeda. 2017, Game [PlayStation 4, Xbox One, Windows] Electronic Arts, Inc.
[19] Eidos Montréal. Shadow of the tomb raider. 2018, Game [PlayStation 4, Xbox One, Windows] Square Enix Co., Ltd.
[20] Volition, Inc. Red faction. 2001, Game [PlayStation 2, GameCube, Xbox, Windows, Macintosh] THQ.
[21] Gabriel Interactive Inc. Construction destruction. 2003, Game [Windows] ValuSoft Inc.
[22] Day 1 Studios. Fracture. 2008, Game [PlayStation 3, Xbox 360] LucasArts.
[23] WeltenbauerSoftware Entwicklung GmbH. Construction simulator 3 - console edition. 2020, Game [PlayStation 4, Xbox One] astragon Entertainment GmbH.
[24] Zagrebelnyj P. Spintires. 2014, Game [Windows] Oovee Game Studios.
[25] Saber Interactive. Mudrunner. 2017, Game [PlayStation 4, Xbox One] Focus Home Interactive.
[26] Rockstar Studios. Red dead redemption 2. 2018, Game [PlayStation 4, Xbox One] Rockstar Games.
[27] Rosa M. GPU pro, destructible volumetric terrain. A K Peters/CRC Press; 2010.
[28] Team17. Worms 3D. 2003, Game [PlayStation 2, GameCube, Xbox, Windows, Macintosh] SEGA.
[29] Mojang. Minecraft. 2011, Game [Windows, Macintosh, Linux] Mojang.
[30] Hello Games. No man's sky. 2016, Game [PlayStation 4, Windows] Hello Games.
[31] System Era Softworks. Astroneer. 2019, Game [Xbox One, Windows] System Era Softworks.
[32] Ghost Ship Games. Deep rock galactic. 2020, Game [Xbox One, Windows] Coffee Stain Publishing.
[33] Stobierski T. Relief terrain pack 3. 2021, Game Engine Tool [Unity].
[34] Amandine Entertainment. Digger PRO. 2021, Game Engine Tool [Unity].
[35] Amandine Entertainment. Ultimate terrains - Voxel terrain engine. 2019, Game Engine Tool [Unity].
[36] Careil V. Voxel plugin. 2020, Game Engine Tool [Unreal].
[37] Pahunov D. Voxeland. 2020, Game Engine Tool [Unity].
[38] Hachenberger P, Kettner L, Mehlhorn K. Boolean operations on 3D selective Nef complexes: Data structure, algorithms, optimized implementation and experiments. Comput. Geom. 2007;38(1):64–99, Special Issue on CGAL.
[39] Douze M, Franco J, Raffin B. QuickCSG: Fast arbitrary boolean combinations of N solids. Research report, 2017, arXiv.
[40] Ju T, Losasso F, Schaefer S, Warren J. Dual contouring of hermite data. In: Proceedings of the 29th annual conference on computer graphics and interactive techniques. 2002, p. 339–46.
[41] Goldfeather J, Molnar S, Turk G, Fuchs H. Near real-time CSG rendering using normalization and geometric pruning. Tech. rep. TR88-006, The University of North Carolina at Chapel Hill. Department of Computer Science; 1988.
[42] Stewart N, Leach G, John S. An improved Z-buffer CSG rendering algorithm. In: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on graphics hardware. New York, NY, USA: ACM; 1998, p. 25–30.
[43] Stewart N, Leach G, John S. Improved CSG rendering using overlap graph subtraction sequences. In: Proceedings of the 1st international conference on computer graphics and interactive techniques in Australasia and South East Asia. New York, NY, USA: ACM; 2003, p. 47–53.
[44] Hable J, Rossignac J. Blister: GPU-based rendering of Boolean combinations of free-form triangulated shapes. ACM Trans Graph 2005;24(3):1024–31.
[45] Romeiro F, Velho L, De Figueiredo LH. Hardware-assisted rendering of csg models. In: 2006 19th Brazilian symposium on computer graphics and image processing. IEEE; 2006, p. 139–46.
[46] Ulyanov D, Bogolepov D, Turlapov V. Interactive vizualization of constructive solid geometry scenes on graphic processors. Program Comput Softw 2017;43(4):258–67.
[47] Zanni C, Claux F, Lefebvre S. HCSG: Hashing for real-time CSG modeling. Proc ACM Comput Graph Interact Tech 2018;1(1):1–19.
[48] Pfister H, van Baar MZJ, Gross M. Surfels: Surface elements as rendering primitives. In: Proceedings of the 27th annual conference on computer graphics and interactive techniques. SIGGRAPH '00, New York, NY, USA: ACM Press/Addison-Wesley Publishing Co.; 2000, p. 335–42.
[49] Shimrat M. Algorithm 112: Position of point relative to polygon. Commun ACM 1962.
[50] Hacker R. Certification of algorithm 112: Position of point relative to polygon. Commun ACM 1962;5(12):606.
[51] Alonso J, Joan-Arinyo R. Back-to-front ordering of triangles in digital terrain models over regular grids. J Comput Sci Tech 2018;33(6):1192.
[52] Real-time footage video of our approach. 2023, https://bit.ly/demoCEIG23.
[53] Cherchi G, Pellacini F, Attene M, Livesu M. Interactive and robust mesh booleans. 2022, arXiv preprint arXiv:2205.14151.
[54] Zhou Q, Grinspun E, Zorin D, Jacobson A. Mesh arrangements for solid geometry. ACM Trans Graph 2016;35(4):1–15.
[55] Jacobson A, Panozzo D. Libigl: Prototyping geometry processing research in c++. In: SIGGRAPH Asia 2017 courses. 2017, p. 1–172.
[56] Tevs A, Ihrke I, Seidel H-P. Maximum mipmaps for fast, accurate, and scalable dynamic height field rendering. In: Proceedings of the 2008 symposium on interactive 3D graphics and games. 2008, p. 183–90.