Fast, Believable Real-time Rendering of Burning Low-Polygon Objects in Video Games

Dhanyu Amarasinghe and Ian Parberry

Technical Report LARC-2011-04

Laboratory for Recreational Computing
Department of Computer Science & Engineering
University of North Texas
Denton, Texas, USA

June 2011

Fast, Believable Real-time Rendering of Burning Low-Polygon Objects in Video Games

Dhanyu Amarasinghe
Dept. of Computer Science & Engineering
University of North Texas
dhanyuamarasinghe@unt.edu

Ian Parberry
Dept. of Computer Science & Engineering
University of North Texas
ian@unt.edu

ABSTRACT

We present a framework for modeling the deformation and consumption of low-polygon models under combustion while generating procedural fire. Many recent publications have shown variety of deformation techniques involved in computer graphics using the GPU. However, deformation of low-polygon models, (which are prevalent in video games) is quite challenging since it is hard to maintain realism. We present a method for emulating the consumption and deformation of low-polygon models using real-time mesh refinement. Our focus is on trading realism for computation speed so that processing power is available for other tasks, such as might arise in the current generation of video game. We have implemented and tested our method on a relatively modest GPU using CUDA. Our experiments suggest that our method gives a believable rendering of the effects of fire while using only a small fraction of CPU and GPU resources. Our method also allows for quick and easy LOD (level-of-detail) rendering of burning objects.

Categories and Subject Descriptors

I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*hierarchy and geometric transformations, animation*; I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*animation*; I.6.8 [**Simulation and Modelling**]: Type of Simulation—*animation*

General Terms

Algorithms, Experimentation

Keywords

Hardware acceleration, volume rendering, freeform deformation, procedural, generation, low-polygon modeling, CUDA, refinement, subdivision.

1. INTRODUCTION

Model deformation is an essential part of maintaining the realism of physical objects in video games. While high quality graphics is an inescapable necessity for the modern video game, developers must choose detailed structure of game models carefully due to the limitations of hardware resources and processing power needed in real time rendering. One of the key features of such detailed structure is a number of polygons per model. Low-polygon models are typically used as much as possible, with their deficits hidden by a choice of pragmatic textures. As a tradeoff between quality and performance, many game developers use extremely low-polygon models for most of the flat surfaces in the game environment such as doors, windows, and walls. Since deformation of such low-polygon models while maintaining realism is quite thorny, developers commonly resort to model swapping techniques.

In this paper we consider real-time emulation of the deformation and consumption of low-polygon models due to combustion. Fire simulations may be used effectively to increase the reality of visual effects in computer animations. Real-time triangle subdivision is a useful technique, but complete subdivision of each and every model is not practical in real time. We extend the method discussed in Amarasinghe and Parberry [2, 4] to introduce a real-time refinement method that can be used in deformation and real-time rendering of burning low-polygon models while maintaining performance and realism. Our aim is to increase believability by a large amount while increasing computation load only minimally.

The structure of the remainder of this paper is as follows. In Section 2 we describe some related work. In Section 3 we describe our representational framework for the subdivision and deformation, and the key features of such a strategy. Section 4 describes our approach to implementing LOD. Section 5 describes the results of a CUDA implementation of our algorithm. Section 6 contains the conclusion and further work. See also our fire web page (Amarasinghe and Parberry [3]) for more images and a video demonstration.

2. PREVIOUS WORK

In Amarasinghe and Parberry [2, 4] we describe a technique for emulating the consumption and deformation of high-polygon models due to fire. The obvious way to extend this technique to low-polygon models is to use real-time mesh refinement, subdividing triangles only when necessary.

There is a great amount of prior work on subdivision surface schemes for fast mesh refinement in real-time applications. Wicke and Ritchie [16] introduce the technique of mesh re-

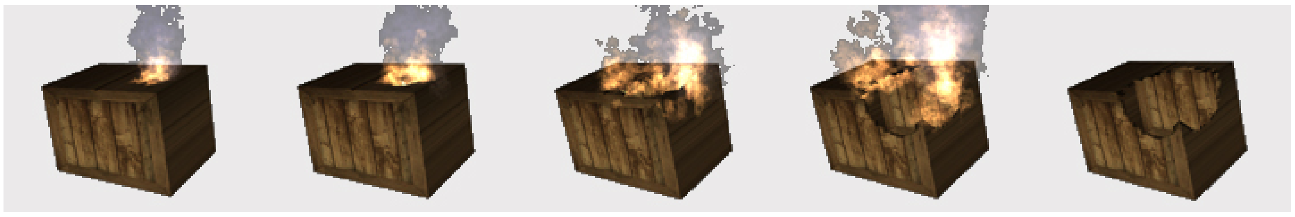
Figure 1: The consumption of a low-polygon model and the spread of procedural fire.

finement to capture detailed physical behavior in simulating fractures by subdividing mesh elements. Hormann and Labsik [8, 11] discuss what kind of parameterizations are optimal for remeshing. Peyré and Cohen [13] describe a method for adaptive mesh refinement for an expanding heat boundary. The papers of Guo and Li [9], and He and Schaefer [10] discuss parametric subdivision of mesh surfaces. Giraud-Moreau, Borouchaki, and Cherouat [5] perform real time deformation by applying remeshing to selective material. Kovacs and Mitchell's crease approximation method [12] also contains useful information about surface subdivision. The survey paper of Alliez and Ucelli [1] on remeshing of surfaces is also quite useful.

Relatively little work has been published about hardware assisted implementation of subdivision schemes. Boubekeur and Schlick [5, 6] introduced useful mesh refinement techniques using modern GPUs. Schive, Tsai, and Tzihong [14] use the GAMeR technique using GPUs for adaptive mesh refinement in astrophysics. Fan and Cheng [7], and Fünfzig and Müller [15] discuss a few techniques for subdivision using modern GPUs.

3. SUBDIVISION

Graphics Processing Units (GPUs) are no longer limited to scene rendering, but have also have been used for general purpose computing. Technology such as CUDA (Compute Unified Device Architecture) developed by NVIDIA provide a platform for implementing general purpose computation on GPUs. However, as mentioned by Boubekeur and Schlick [6], there are limitations to data translation from CPU to GPU, since current graphics hardware is unable to generate more polygons than those sent through the graphics bus by the application running on the CPU. Consequently, we have adapted Boubekeur and Schlick's [6] Generic Adaptive Mesh Refinement (GAMeR) technique to procedurally create additional inner vertices on-the-fly.

This section is divided into three subsections. The first subsection describes the mesh refinement process and its properties. The second section introduces the concept of *eligible* triangles using barycentric coordinates. The third subsection shows how these can be used to perform triangle deformation.

3.1 Refinement Patterns and Properties

Although our approach is valid for other polygonal shapes, we only consider the case of triangular meshes in this paper, since that is what is primarily used in video games. We pre-compute all of the useful refinement configurations

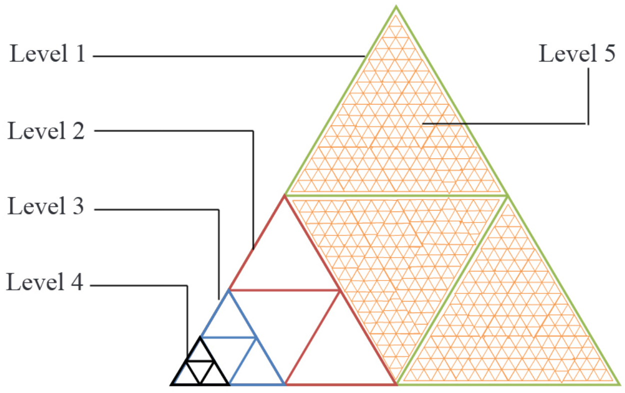

Figure 2: The Adaptive Refinement Pattern showing the level subdivisions for a single triangle.

of a single triangle using a technique called *uniform decomposition*, in which the subdivision takes place in all of the cells recursively. We use an isotropic template that divides each triangle into half for five recursive levels in depth as illustrated in Figure 2. This resulting Adaptive Refinement Pattern (ARP for short) is stored once on the GPU as a vertex buffer object.

Recall that our objective is not to subdivide each and every triangle in the object. Our aim is to subdivide only when necessary, and prior to deformation. We declare some attributes for each of the subdivided triangles in Table 1. One of the important attributes in this set is the *status* of the triangle. The values -1, 0, 1, 2 represent the triangle's status as *inactive*, *initial*, *active*, and *processed* respectively. The renderer will draw only the final ARP of *active* and *processed* triangles generated by each coarse triangle. We will discuss the rest of these attributes along with our deformation algorithm in Section 3.3.

After loading the ARP and its attributes to the vertex buffer, we need to map ARP coordinates to the corresponding coarse polygon using a displacement map similar to Figure 2. Unlike Boubekeur and Schlick [6], we record the final coordinate set into the GPU since we have yet to deform our vertices prior to rendering. At this point we apply ARP to the coarse polygon only if it is eligible to proceed to the next level of subdivision. This eligibility depends upon the location of the heat boundary relative to the triangle.

Attribute	Values	Description
id	Integer	Track siblings/parent
SetLevel	$1, 2, 3, 4, 5$	Depth of the division
Siblings	Integer	Number of siblings
Parent	Integer	Parent id
Status	$-1, 0, 1, 2$	Status of the triangle

Table 1: ARP attributes.

3.2 Barycentric Points & the Heat Boundary

The temperature of a burning object in the real world changes over both time and space. Temperature increase due to combustion influences the mechanical behavior of the object, and the thermal conductivity of the object influences the thermal response.

To speed up computation, we approximate the expansion of the heat boundary by calculating it around a single fixed point, following our heat boundary model described in Amarasinghe and Parberry [2, 4]. The approximated heat boundary expansion is given by:

$$\begin{aligned} R^2 &= |\sin(\pi\Theta/\Delta r) + \sin(\pi\Theta) + \\ &\quad \psi((x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2)|, \end{aligned}$$

where $R = r+\Delta r$, the radius r incremented by Δr in each Δt time period. The angle Θ is a random value that makes the expanding heat boundary irregular in shape. The location of the heat source is (x_0, y_0, z_0). However, the value of heat index constant ψ from Amarasinghe and Parberry [2, 4], which is supposed to be a constant that depends only on the size of the coarse triangles of the model, is no longer fixed. Therefore, we let the designer set ψ depending on how many levels of subdivision are planned.

It remains to decide which coarse triangles are eligible for subdivision. This has to be a function of the expanding heat boundary. Furthermore, the subdivision has to take place prior to the deformation process. Our solution is to send a virtual heat wave through the model prior to the actual heat boundary expansion. This creates an area in addition to the three initial heat boundary areas described in Figure 3 of Amarasinghe and Parberry [2]. Since the introduced boundary expansion takes place prior to the three original expanding boundaries (see Figure 3), we can proceed with the subdivision of qualified triangles before the deformation process begins.

Since we are using a single source heat boundary, temperature at all points will depend on the distance from the heat source at (x_0, y_0, z_0). If this point is in the middle of one of the coarse triangles, the triangle will not be eligible for subdivision until the virtual heat boundary hits one of its vertices. To avoid such issues we represent each triangle using barycentric coordinates as follows.

Suppose point $P = (x, y, z)$ is given by:

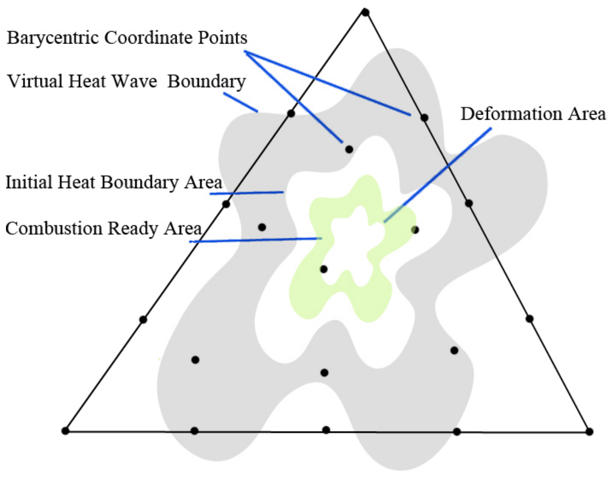
Barycentric Coordinate Points
Virtual Heat Wave Boundary
Deformation Area
Initial Heat Boundary Area
Combustion Ready Area

Figure 3: Heat boundary areas and barycentric point sets.

$$\begin{aligned} x &= \lambda_1 x_1 + \lambda_2 x_2 + \lambda_3 x_3 \\ y &= \lambda_1 y_1 + \lambda_2 y_2 + \lambda_3 y_3 \\ z &= \lambda_1 z_1 + \lambda_2 z_2 + \lambda_3 z_3, \end{aligned}$$

where λ_1,λ_2 and λ_3 are area parameters such that

$$\lambda_1 + \lambda_2 + \lambda_3 = 1$$

.

We need to calculate the barycentric coordinates for non-eligible triangles only, where *eligible* triangles are those that are close to the heat boundary. The following algorithm returns `true` if coarse triangle T is eligible.

```
for each coarse triangle $T$
  if $T$ is not eligible then get barycentric point set $P$

for each barycentric point set $P$
  if $P$ is inside the heat boundary
    return true
```

3.3 Deformation

After applying ARP to the eligible triangle, we next apply deformation techniques. Although our ARP arbitrarily contains triangles of five levels in depth (see Figure 2), this number can be changed in the obvious fashion by the designer. Deformation applies only to the final level (in our case, fifth level) status *active* triangles. At this point, rendering all levels of triangles in the ARP will be costly and wasteful. Instead, we choose which triangles to render using the ARP attributes listed in Table 1.

The process can be described informally as follows. Initially, the coarse triangles of the model are considered *active* (status value 1) triangles, and all ARP triangles are initialized as *initial* (status value 0) triangles. Each subdivided triangle consists of three siblings and a parent. As our algorithm

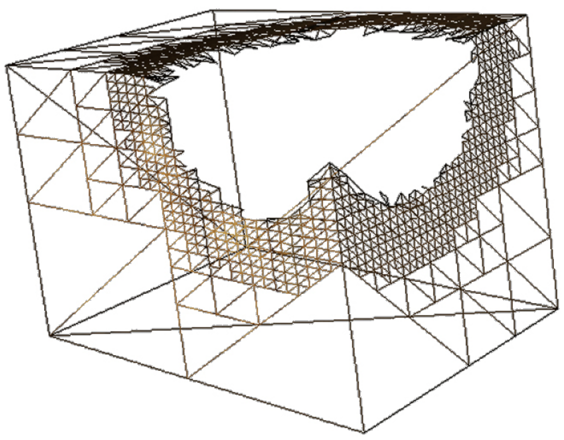

Figure 4: The refinement hierarchy and deformation applied to a 12-triangle model of a box.

proceeds, if one of the child triangles turns *active*, then the parent will turn *processed* (status value 2) until all of its children also become status *active*. Once its children have all turned *active*, the parent triangle will change its status from *processed* to *inactive* (from status value 2 to -1). More specifically:

```
if triangle has SetLevel = 5 and Status = 0
   if triangle is inside the heat boundary
      Status := 1
      Status of all siblings := 1
      Status of parent := 2

if SetLevel <5 and Status >0
   if sibling's Status >-1
      sibling Status := 1
      parent Status := 2

if all children have Status = 1 and parent has Status = 2
   parent Status := -1
```

In our high-polygon deformation algorithm (Amarasinghe and Parberry [2, 4]), the displacement of each vertex depends on the surrounding vertices. Therefore, to apply proper calculation of deformation, we must let the subdivision proceed a few steps further before applying deformation to the mesh. By doing so, we are able to calculate the proper strength factors within the deforming triangles properly. Figure 4 illustrates the refinement hierarchy and deformation applied to a low-polygon model of a box.

4. LEVEL SETS AND DISTANCE

In a game environment, objects located far from the viewer need not be rendered in as much fine detail as those close up. A significant speed-up can be obtained by having models stored at various *levels of detail* (abbreviated LOD) ranging from, for example, hundreds of triangles for objects in the far distance to tens of thousands for close-up objects. These variants of the model are usually created by the artist, although procedural methods do exist.

Polygon Count	Fully Subdivided	Our Method	Speed-up Factor
10k	84fps	165fps	1.96
15k	76fps	159fps	2.09
20k	63fps	153fps	2.43
50k	48fps	60fps	1.25

Table 2: Frame rate of fully subdivided model versus our approach.

Our algorithm allows us to implement LOD for burning objects by controlling the level of adaptive refinement of the coarse mesh triangles. We calculate the distance between object and the player in the CPU and pass it to the GPU as a parameter. Level adjustment is decided and passed to the appropriate ARP before rendering.

For a solid object the level of refinement is directly proportional to the distance. However, surface removal and deformation of a burning object makes it slightly more challenging to maintain a smooth transition between level swaps. Define the *burn level* of a model as the number of triangles of the model that have been consumed by fire (after Amarasinghe and Parberry [2, 4]). We then use the following algorithm to determine whether to render triangle T.

```
if number of children of $T$ with
higher burn level than $T$ is $\geq 2$
and SetLevel $\geq 5$
   then hide $T$
   else show $T$
```

Figure 5 illustrates our burning box model at different LODs.

5. EXPERIMENTS

The images of a burning box shown in this paper are screenshots from a CUDA implementation of our algorithm applied to a model with 12 triangles. The flames are generated using 2000 fire particles and 500 smoke particles. The advantage of such a system is clear when comparing the resources required to deform a completely subdivided model versus deforming a low-polygon model using our method. Table 2 shows the frame rates of the animation when our algorithm is implemented in CUDA on relatively modest hardware; An Intel®Core$^{\text{TM}}$2 Duo CPU P8400 @ 2.26GHz processor with an NVidia GeForce 9800 GTS graphics card. This performance will of course be much better on the current generation of graphics hardware, but that is not our aim. Our aim is to provide detail sufficient to trigger willing suspension of disbelief at a relatively low cost in computation load.

The outcome of these experiments shows that our method results in doubling the frame rate. Therefore, we believe this approach is a better alternative than subdividing the complete model when it comes to deforming low-polygon models.

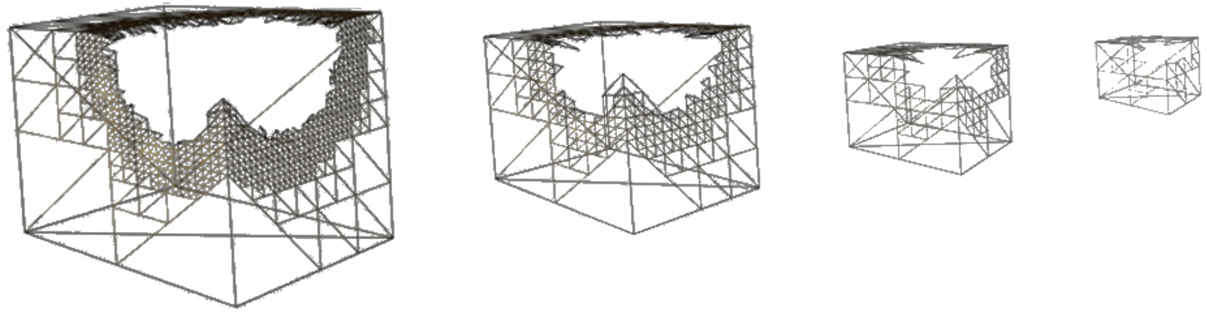

Figure 5: Real-time computation of LOD for a burning object.

6. CONCLUSION AND FUTURE WORK

We have proposed a method for the real-time deformation and consumption of a low-polygon model during combustion by procedurally generated fire. By doing so, we have extended our work in Amarasinghe and Parberry [2, 4] to low-polygon models. We have performed simulation of real-time deformation and consumption of any model regardless of the size of the triangles. We have focused on the performance with a reasonable amount of realism sufficient to trigger willing suspense of disbelief in the game player. Our simulations have performed well on a model with low-polygon count and large triangles.

Most of the models used in video games appear to be shell models, with a hollow interior. Deformation of shell models is different from solid models. We intend to investigate the extension of our method to solid models, and to investgate a better approximation to heat boundary expansion.

7. REFERENCES

[1] P. Alliez, G. Ucelli, C. Gotsman, and M. Attene. Recent advances in remeshing of surfaces. *Shape Analysis and Structuring*, pages 53–82, 2008.

[2] D. Amarasinghe and I. Parberry. Towards fast, believable real-time rendering of burning objects in video games. Technical Report LARC–2010–04, Laboratory for Recreational Computing, Dept. of Computer Science & Engineering, Univ. of North Texas, October 2010.

[3] D. Amarasinghe and I. Parberry. Fire reloaded, `http://larc.unt.edu/ian/research/fire2/`, 2011.

[4] D. Amarasinghe and I. Parberry. Towards fast, believable real-time rendering of burning objects in video games. In *Proceedings of the 6th Annual International Conference on the Foundations of Digital Games*, 2011.

[5] H. Borouchaki, P. Laug, A. Cherouat, and K. Saanouni. Adaptive remeshing in large plastic strain with damage. *International Journal for Numerical Methods in Engineering*, 63(1):1–36, 2005.

[6] T. Boubekeur and C. Schlick. Generic mesh refinement on GPU. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, pages 99–104. ACM, 2005.

[7] F. Fan and F. Cheng. GPU supported patch-based tessellation for dual subdivision. In *2009 Sixth International Conference on Computer Graphics, Imaging and Visualization*, pages 5–10. IEEE, 2009.

[8] M. Floater and K. Hormann. Surface parameterization: A tutorial and survey. *Advances in Multiresolution for Geometric Modelling*, pages 157–186, 2005.

[9] X. Guo, X. Li, Y. Bao, X. Gu, and H. Qin. Meshless thin-shell simulation based on global conformal parameterization. *IEEE Transactions on Visualization and Computer Graphics*, pages 375–385, 2006.

[10] L. He, S. Schaefer, and K. Hormann. Parameterizing subdivision surfaces. *ACM Transactions on Graphics*, 29(4):1–6, 2010.

[11] K. Hormann, U. Labsik, and G. Greiner. Remeshing triangulated surfaces with optimal parameterizations. *Computer-Aided Design*, 33(11):779–788, 2001.

[12] D. Kovacs, J. Mitchell, S. Drone, and D. Zorin. Real-time creased approximate subdivision surfaces. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games*, pages 155–160. ACM, 2009.

[13] G. Peyré and L. Cohen. Geodesic remeshing using front propagation. *International Journal of Computer Vision*, 69(1):145–156, 2006.

[14] H. Schive, Y. Tsai, and T. Chiueh. Gamer: A graphic processing unit accelerated adaptive-mesh-refinement code for astrophysics. *The Astrophysical Journal Supplement Series*, 186:457, 2010.

[15] V. Settgast, K. Müller, C. Fünfzig, and D. Fellner. Adaptive tesselation of subdivision surfaces. *Computers & Graphics*, 28(1):73–78, 2004.

[16] M. Wicke, D. Ritchie, B. Klingner, S. Burke, J. Shewchuk, and J. O'Brien. Dynamic local remeshing for elastoplastic simulation. *ACM Transactions on Graphics*, 29(4):1–11, 2010.