Proceedings of the 6th Annual North American Conference on AI and Simulation in Games (GAMEON-NA), pp. 21-26, EUROSIS, Troy, NY, 2011.

FAST, BELIEVABLE REAL-TIME RENDERING OF BURNING LOW-POLYGON OBJECTS IN VIDEO GAMES

Dhanyu Amarasinghe and Ian Parberry
Department of Computer Science & Engineering
University of North Texas
Denton, TX, USA
Email: dhanyu@gmail.com, ian@unt.edu

KEYWORDS
Hardware acceleration, deformation, procedural content generation, low-polygon modeling, CUDA, GPU.

ABSTRACT

Deformation of the low-polygon models used in video games is challenging since it is hard to maintain realism. We show how real-time mesh refinement can be used for modeling the deformation and consumption of low-polygon models under combustion while generating procedural fire. Our focus is on trading realism for computation speed so that processing power is still available for other computational tasks. Our method also allows for quick and easy LOD (level-of-detail) rendering of burning objects. We have implemented and tested our method on a relatively modest GPU (Graphics Processing Unit) using NVIDIA's CUDA (Compute Unified Device Architecture). 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.

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.

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 our prior work (Amarasinghe and Parberry 2011b) 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.

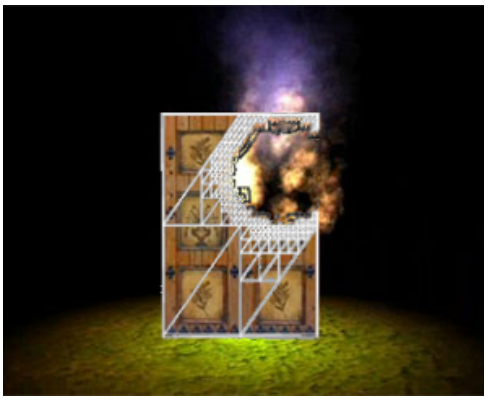

Figure 1: Procedural triangulation of a burning door.

We are able to triangulate burning low-polygon objects on-the-fly in response to real-time procedural fire in order to provide more detail where it is needed Figure 1 shows where a simple 12-triangle door is triangulated near the source of compbustion, and Figure 2 show a 12-triangle box at various stages of burning. The interested reader can visit our fire web page (Amarasinghe and Parberry 2011a) for larger color images and a video demonstration of a burning low-polygon door as shown in Figure 1.

The remainder of this paper is divided into four main sections. The first section sets the context for this paper by describing prior work. The second section describes our triangle subdivision algorithm. The third section shows that our method is particularly suited to producing models at different levels of detail for faster rendering. The fourth section describes the results of some preliminary experiments with a CUDA implementation of our algorithm.

Figure 2: The consumption of a low-polygon model and the spread of procedural fire.

PRIOR WORK

In (Amarasinghe and Parberry 2011b) 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. The idea of using mesh refinement was used in (Wicke et al. 2010) to capture detailed physical behavior in simulating fractures by subdividing mesh elements. The kind of parameterizations that are optimal for remeshing are discussed in (Floater and Hormann 2005, Hormann et al. 2001). A method for adaptive mesh refinement for an expanding heat boundary is discussed in (Peyré and Cohen 2006). The papers (Guo et al. 2006), and (He et al. 2010) discuss parametric subdivision of mesh surfaces, while (Borouchaki et al. 2005) performs real time deformation by applying remeshing to selective material. Useful information about surface subdivision can be found from Kovacs and Mitchell's crease approximation method (Kovacs et al. 2009). The survey paper (Alliez et al. 2008) on remeshing of surfaces is also quite useful.

Relatively little work has been published about hardware assisted implementation of subdivision schemes. Some useful mesh refinement techniques using modern GPUs can be found in (Borouchaki et al. 2005, Boubekeur and Schlick 2005). The so-called GAMeR technique from (Schive et al. 2010) uses the GPU for adaptive mesh refinement in astrophysics. Some useful techniques for subdivision using modern GPUs can be found in (Fan and Cheng 2009) and (Settgast et al. 2004).

SUBDIVISION

Graphics Processing Units (GPUs) are no longer limited to just scene rendering. Technology such as NVIDIA's CUDA (Compute Unified Device Architecture) provides a platform for implementing general purpose computation on GPUs. However, as mentioned in (Boubekeur and Schlick 2005), 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 their Generic Adaptive Mesh Refinement (GAMeR) technique to procedurally create additional inner vertices on-the-fly.

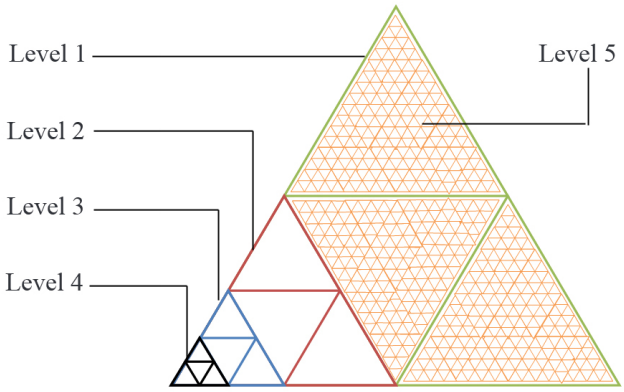

Figure 3: The Adaptive Refinement Pattern showing the level subdivisions for a single triangle.

The remainder of this section is divided into three subsections. The first subsection discusses refinement patterns and properties. The second subsection discusses the use of barycentric points in relation to the heat boundary. The third subsection brings these concepts together in our deformation algorithm.

Refinement Patterns and Properties

Although our approach is valid for other polygonal shapes, we only consider the case of triangular meshes, since that is what is primarily used in video games. We pre-compute all of the useful refinement configurations 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 3. 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

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.

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.

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 3. Unlike (Boubekeur and Schlick 2005), 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.

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 2011b). The approximated heat boundary expansion is given by:

$$R^2 = |\sin(\pi\Theta/\Delta r) + \sin(\pi\Theta) + \psi((x-x_0)^2 + (y-y_0)^2 + (z-z_0)^2)|,$$

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 2011b), 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 2010). Since the introduced boundary expansion takes place prior to the three original expanding boundaries (see Figure 4), 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:

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

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`

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 3), 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 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

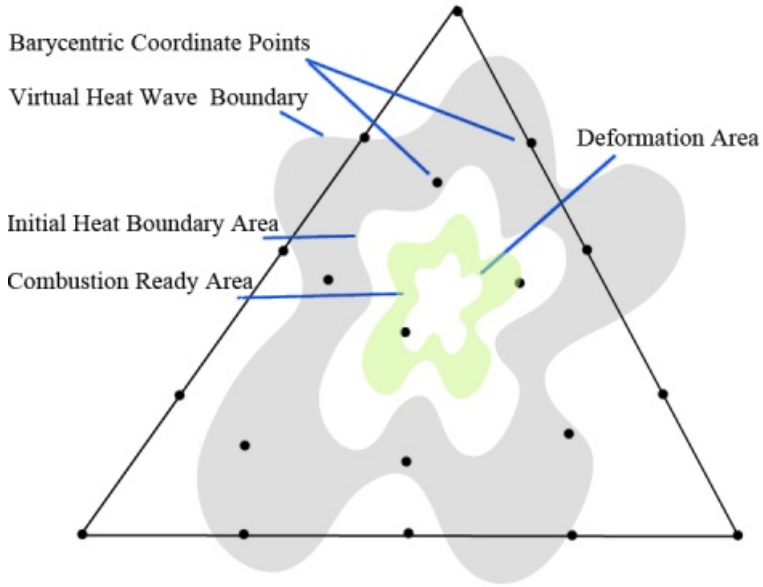

Figure 4: Heat boundary areas and barycentric point sets.

```
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
```

Figure 5: Algorithm for computing Status.

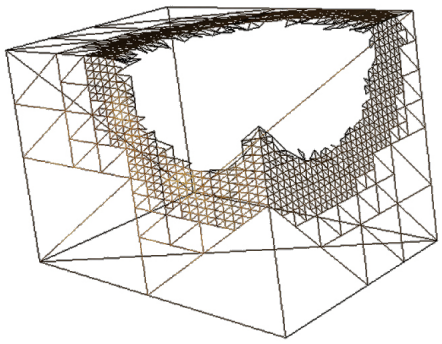

Figure 6: The refinement hierarchy and deformation applied to a 12-triangle model of a box.

processed to *inactive* (from status value 2 to -1). See Figure 5 for the details.

In our high-polygon deformation algorithm (Amarasinghe and Parberry 2011b) 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 6 illustrates the refinement hierarchy and deformation applied to a low-polygon model of a box.

LEVEL OF DETAIL

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.

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

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.

the *burn level* of a model as the number of triangles of the model that have been consumed by fire as in (Amarasinghe and Parberry 2011b). We then use the following algorithm to determine whether to render triangle T: Show T if and only if the number of children of T with higher burn level than T is ≥ 2, and SetLevel ≥ 5. Figure 7 shows our burning box at different LODs.

EXPERIMENTS

The images of burning objects shown here and in (Amarasinghe and Parberry 2011a) 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™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.

CONCLUSION AND FURTHER 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 2011b) 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. We intend to investigate the extension of our method to solid models, and to investgate a better approximation to heat boundary expansion.

REFERENCES

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

Amarasinghe D. and Parberry I., 2010. *Towards Fast, Believable Real-time Rendering of Burning Objects in Video Games*. Tech. Rep. LARC–2010–04, Laboratory for Recreational Computing, Dept. of Computer Science & Engineering, Univ. of North Texas.

Amarasinghe D. and Parberry I., 2011a. *Fire Reloaded*. URL `http://larc.unt.edu/ian/research/fire2/`.

Amarasinghe D. and Parberry I., 2011b. *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*. 256–258.

Borouchaki H.; Laug P.; Cherouat A.; and Saanouni K., 2005. *Adaptive remeshing in large plastic strain with damage*. *International Journal for Numerical Methods in Engineering*, 63, no. 1, 1–36.

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

Fan F. and Cheng F., 2009. *GPU Supported Patch-Based Tessellation for Dual Subdivision*. In *2009 Sixth International Conference on Computer Graphics, Imaging and Visualization*. IEEE, 5–10.

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

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

He L.; Schaefer S.; and Hormann K., 2010. *Parameterizing subdivision surfaces*. *ACM Transactions on Graphics*, 29, no. 4, 1–6.

Hormann K.; Labsik U.; and Greiner G., 2001. *Remeshing triangulated surfaces with optimal parameterizations*. *Computer-Aided Design*, 33, no. 11, 779–788.

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

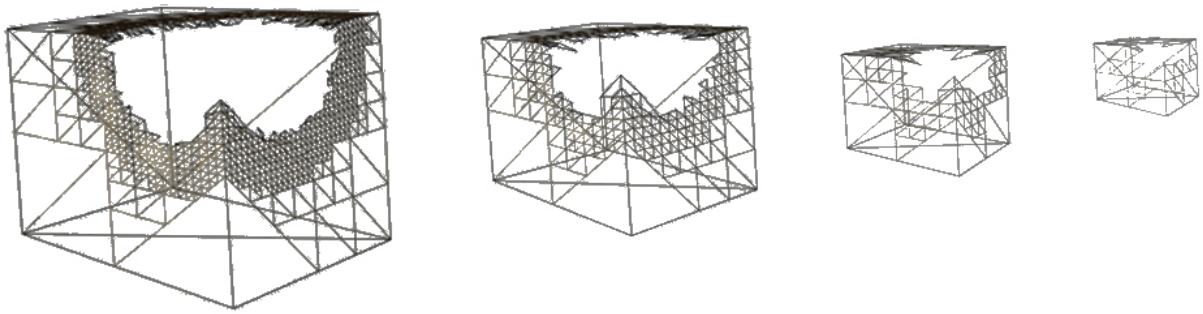

Figure 7: Levels of detail for a burning object provided by our method.

Peyré G. and Cohen L., 2006. *Geodesic remeshing using front propagation*. *International Journal of Computer Vision*, 69, no. 1, 145–156.

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

Settgast V.; Müller K.; Fünfzig C.; and Fellner D., 2004. *Adaptive tesselation of subdivision surfaces*. *Computers & Graphics*, 28, no. 1, 73–78.

Wicke M.; Ritchie D.; Klingner B.; Burke S.; Shewchuk J.; and O'Brien J., 2010. *Dynamic local remeshing for elastoplastic simulation*. *ACM Transactions on Graphics*, 29, no. 4, 1–11.

BIOGRAPHY

DHANYU AMARASINGHE is a native of Sri Lanka. He is currently a PhD student in the Department of Computer Science and Engineering at the University of North Texas. His research interests include graphics for game development, particularly the real-time deformation and consumption of virtual objects by procedural fire. He can be contacted at `dhanyu@gmail.com`. His home page is `http://dhanyu.com/`.

IAN PARBERRY was born in London, England and emigrated as a child with his parents to Brisbane, Australia. After obtaining his undergraduate degree there from the University of Queensland he returned to England for a PhD from the University of Warwick. He has worked in academia in the US ever since. He is currently a full Professor in the Department of Computer Science and Engineering at the University of North Texas where he recently stepped down from a 2-year term as Interim Department Chair. A pioneer of the academic study of game development since 1993, his undergraduate game development program was ranked in the top 50 out of 500 in North America by The Princeton Review in 2010. He is on the Editorial Boards of the *Journal of Game Design and Development Education*, *IEEE Transactions on Computational Intelligence and AI in Games*, and *Entertainment Computing*, and he serves as the Secretary of the Society for the Advancement of the Science of Digital Games, which organizes the Annual Foundations of Digital Games conference. He is the author of 6 books and over 80 articles over 30 years' experience in academic research and education. His *h*-index is 18 and his Erdös number is 3. He can be contacted at `ian@unt.edu` or on Facebook. His home page is `http://larc.unt.edu/ian`.