CS6610 Term Project
Billboard Clouds for Extreme Model Simplification

Junxing Zhang and Guodong Li
{junxing, ligd}@cs.utah.edu

Goal
We have achieved the goal in our project proposal. That is to implement a greedy optimization procedure for constructing billboard clouds in an error-based approach using an object-space error metric. We produced an openGL program that renders 3D objects with “billboard clouds”, namely using the greedy optimization that respects the “density” threshold to construct a set of textured, partially transparent polygons (or billboards) [Xavier Decoret, Fredo Durand, Francois Sillion, and Julie Dorsey. Billboard clouds for extreme model simplification. In Proceedings of the ACM Siggraph. ACM Press, 2003]. The maximum tolerable error is set and the program uses this value to build the billboards with minimum costs. We showed some examples of using billboard cloud to simplify the modeling of complex 3D objects (e.g. OFF objects).

Billboard clouds do not suffer from deformation, resolution mismatch, and rubber sheet effects. It also completely alleviate the incomplete representation issue since our image-based representation is constructed in object space and represents all triangles. The cracks are the main artifact that are not explicitly prevented by the method. In order to reduce crack artifacts, the triangles should be projected on multiple planes (instead of just be collapsed into one plane). The authors also claim that the crack artifacts could be reduced by redistributing triangles and re-optimizing plane locations in a final relaxation step. Billboard clouds are effective in simplifying complex models with multiple textures into a small number of textured polygons. They maintain precise silhouettes as well as interesting parallax effects even into the range of extreme simplification.

Results
A space shuttle with about 700 faces (triangles) are rendered as billboard clouds. The following four pictures show the shuttle rendered with error threshold set to 0.1, which results in 120 billboards. For comparison, the shuttle is also rendered in the original way (as triangles) by the side of the billboard clouds. Note that the original object and the corresponding billboard cloud are not perfectly aligned since we rotate them and change the viewpoint to compare them in a convenient way.
With less billboards, we could speed up the rendering while maintaining acceptable visual quality. The following first two pictures are rendered with the error threshold set to 0.3, which results in 66 billboards. The second two pictures are rendered with the error threshold set to 0.5, which results in 11 billboards.
The textures for the above 11-billboard cloud are shown side by side in the image below:
**Implementation**

We took a series of loops to get the billboards. Each billboard will have an orientation defined in terms of theta and phi. A distance rho from a defined origin will also be iterated. For each billboard the program verifies for each triangle in the model if this triangle is sufficiently close to the billboard by checking the predefined error threshold. If so, the triangle is valid and the area of the polygon projected on the billboard is calculated and accumulated into the corresponding bin. Then, all the areas of each of the valid triangles are added and stored in an array of billboards. Once all possible billboard orientations have been taken into account, the whole billboard array is swept to find the billboard with the most coverage. Eventually all polygons are valid for at least one billboard. After all billboards have been calculated, the min and max values of the sizes of the billboards are calculated by getting the highest and lowest x and y values. Then we get the size of each billboard.

All of the valid faces for each billboard are drawn and projected orthogonally (using the min and max values as projection sizes) to generate the textures. Then, a quad object with the min and max values as its vertices and the normal of the billboard as the orientation position and rho as a translation is generated and textured.

**Projected Area calculation:** In order to calculate the coverage or penalty of a face with regard to a plane, the projected area of the face on the plane must be obtained first. Because we know the coordinates of three vertices of the face, it is convenient to use Heron's Formula for triangle area. It is stated as:

\[
\text{area} = \sqrt{s(\text{length}_{\text{ab}})(\text{length}_{\text{bc}})(\text{length}_{\text{ca}})}
\]

where \(\text{length}_{\text{ab}}, \text{length}_{\text{bc}}\) and \(\text{length}_{\text{ca}}\) are side lengths and \(s = \frac{\text{length}_{\text{ab}} + \text{length}_{\text{bc}} + \text{length}_{\text{ca}}}{2}\).

**Projected Point Calculation:** Vertices of faces that collapse to a plane must be projected to the plane before the minimum rectangle that bounds the projected points can be found. The bounding rectangle will then be used to shoot the texture of the billboard that corresponds to the plane. It is important to note that how a point is projected to a plane depends on the relative position of the point to the plane. If a point \(A\) is at the normal side of the plane, the coordinate of the projected point \(A'\) can be found as a vector \(\text{OA}'\), which is obtained by subtract the vector \(d\text{N}\) from the vector \(\text{OA}\), where \(O\) is the origin, \(\text{N}\) is the plane normal and \(d\) is the distance from \(A\) to the plane. If \(A\) is at the other side of the plane, the coordinate of the projected point \(A'\) can also be found as the vector \(\text{OA}'\). But in this case \(\text{OA}'\) equals to the addition of \(\text{OA}\) and \(d\text{N}\). To determine which side of the plane a point is located, we find a point in the plane and calculate a vector from this plane point to the point \(A\). Because any vector that points from the plane to its normal side has an angle less than 90 degree with the plane normal, we can use the sign of the dot product between this vector and the normal to determine if the point \(A\) is at the normal side.

**Bounding Rectangle Calculation:** The calculation of the convex hull of two dimensional points is a well-known geometry problem and many algorithms have been proposed to solve this problem. The paper we implemented suggests to use a computational geometry library called CGAL (www.cgal.org). We studied and examined
this library, and found that we don't have time to port code from this library. Instead, we designed our own routine to calculate the convex hull. First, we rotate the normal of the projection plane to one of the coordinate axis using the transformation matrix we learned from the planar reflection lecture. The rotation transform the three dimension points into two dimension points by making all points have the same x, or y, or z value. For the other two coordinate values that vary among points, we find their minimum and maximum values and combine them into four points that define the bounding rectangle.

**Texture Generation:** We shoot billboard textures by rendering faces using an orthographic projection as suggested by the paper. The viewing volume is defined by the bounding rectangle and the near and far clipping planes that are distant enough to enclose all collapsed faces in both sides of the projection plane. Here we use the same technique for texture generation as we used in the cube map project. To avoid “cracks” between adjacent billboards, we tried to render the entire scene instead of just collapsed faces when shooting textures. But we find that this approach introduces another problem while solving the “crack” problem. Namely, the faces that don't collapse to a plane might be shoot into the texture of the plane billboard.

**Texture Rendering:** In rendering the textured billboards, we have to solve another problem. That is we must find the corresponding relationship between the texture coordinates and the vertex coordinates. When shooting textures we only know the plane normal and up vector. Now we have to find the counter-clockwise vertex coordinates starting from the lower left corner based on these info. For proper combination of the billboards and background, we use blending, alpha testing and a black background.

**Problems Encountered In Implementation**

1. In order to decide whether a triangle is in a bin/subbin, each vertex of the triangle is projected on each of the four directions delimiting the bin (four direction vectors surround the center). If the projection of a vertex is among the valid range [min_rh0, max_rh0] of any of the four directions, then this vertex intersects the bin in terms of the given error threshold. The triangle intersects the bin if and only if all its three vertices intersect the bin. However, in the refining we have to divide a bin with the highest density (parent bin) into $6\times6\times6$ subbins, and then attempt to find the subbin with the highest density. It is possible that none of these subbins intersects with any of them in the valid face list of the parent bin since the vertices of the parent bin may intersect exclusively different direction of the parent bin, in this case this triangle may not intersect any of the subbins). Thus, when the valid list of subbin is empty, we conclude that above problem appear, then we choose the first uncollapsed face as the billboard and collapse all valid faces on this billboard. Fortunately, this happen very infrequently and our remedy approach won't hurt the performance much.

2. After a billboard is identified, all relevant faces are collapsed on it. When the materials and textures of these faces are different, the projects of the faces will interleave with each other and jagging appears. Hence, we draw the original objects and take its image in the plane space by look at the reverse direction of the plane normal, then we make the texture from the images and then texture the plane. This method eliminates the interference of the faces on one side of the plane, but still suffers from the jagging
artifact resulting from the projects of the faces on the two sides of the plane. Fortunately, when the number of billboards is large, this artifact becomes insignificant. And, unless we modify the definition of validity (and thus the greedy algorithm), we couldn't eliminate this artifact completely.

3. We use wavefront 3D objects as object models. We ever made mistakes on scaling the objects and obtain incorrectly transformed object models. After we fix it we are able to render the whole objects or parts of them in the exact way the glm library does.

**Possible Extension**

1. Since in this paper a greedy algorithm is used to find the planes on which polygons collapse, and this algorithm isn't guaranteed to be optimal, we may figure out our own greedy algorithm and compare it with the original one. Intuitively, the uniform discretization this paper adopts has some advantages in the aspects of simplification, but it suffer from low performance (has to search all the parameter space) and low accuracy (each bin has the same weight). The authors did propose a optimization of the greedy algorithm: they restrict the valid list of a bin to a maximal compact subset, then analyze the validity set into clusters and use only the cluster with the highest density. A bucket-like partitioning algorithm is used for this task. However, we believe that there are better greedy strategies we could use to find the ideal plane set.

2. Extend the approach to the construction of view-dependent billboard clouds, where the simplification is valid for a restricted region of space. This will facilitate higher levels of simplification.

3. We may also attempt to find more applications for billboard clouds such as collision detection and soft shadows in which planar billboard clouds could approximate complex scenes. We may need to add more parameters and greedy strategies for specific applications.

4. We will try to fix the problem of jagging textures of the billboards. We may use a simple greedy packing algorithm to store all the textures of an entire billboard cloud into a single texture so as to reduce memory usage and text switch. We may also compute the normal maps together with the textures so that a billboard cloud cab be relit in real-time using bump mapping and pixel shaders.

Given one or two weeks, we could implement one or two of above extensions.

**Run-time Interface**

The following shows run-time interface of our programs after the billboard cloud is obtained. Before the billboard cloud is rendered, it takes two or three minutes on the computation of the billboards.