

Silhouette Tracking

Tom Hall (tomhall@bytegeistsoftware.com)

Website: <http://www.bytegeist.com/>

Introduction

Silhouettes are useful in computer graphics for a number of techniques including volumetric shadows, outline rendering, toon shading, and silhouette clipping. However, traditional approaches to find all silhouettes for a model are generally computationally expensive.

This paper describes an adaptation to Markosian's Randomized silhouette detection method [1] that takes advantage of the spatial and temporal coherence of silhouettes. Unlike Markosian's method, this technique is capable of detecting new silhouettes being formed from one frame to the next.

The standard brute force approach to retrieve all silhouettes for a given model is to check every edge every frame. This has a running cost of $O(n)$ where n is the number of edges in the model. For dense models, finding silhouettes this way dramatically increases rendering time.

The Silhouette Tracking technique outlined in this document details a more efficient method of extracting silhouettes for dense meshes than the brute force approach. By using silhouette information from the previous frame, we can avoid checking all edges of a model in the next calculation. The process in brief is

1. Create an edge connectivity table (once only – see preparation below).
2. Perform the first iteration using the standard brute force technique (once only).
3. Track the changes in existing silhouettes using the previous silhouette information.
4. Detect the formation of new silhouettes.

Perform steps 1 and 2 for the first frame only (or in setup). Use steps 3 and 4 to retrieve the silhouette information for each subsequent frame.

There are three main advantages to choosing silhouette tracking over the normal approach. This method:

- Performs much better for dense models (see performance results below).
- Returns an organised list of distinct, ordered silhouettes.
- Can be tuned at runtime for either speed or accuracy.

Definitions

A polygon is front facing (or visible) to a given point P if its plane equation is greater than 0 when P is substituted into it, otherwise it is back facing.

A silhouette edge is an edge that is adjacent to both a front facing polygon and

- A back facing polygon, or
- No face at all (a border edge).

Restrictions

Models must have no edge shared by more than 2 polygons. Silhouette tracking tends to perform slightly better if the model is closed (i.e. no border edges can be found).

Preparation

Silhouette tracking relies on an edge connectivity table. This table maps every polygon in the model to its directly adjacent polygons. The polygons plane equation is also kept. Every polygon consists of exactly 3 vertices.

Figure 1

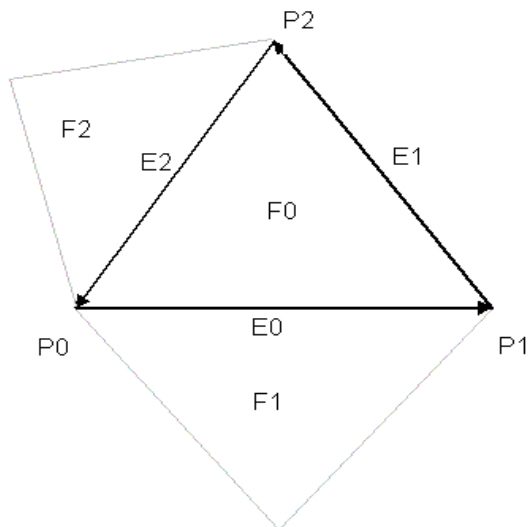


Figure 1 shows an example of a counter-clockwise wound face and its adjacent faces. Note that edge E1 is a border edge. Table 1 shows the entry in the connectivity table for face F0.

Table 1

Face	Edge 0	Edge 1	Edge 2
F0	F1	-	F2

Keep a list of all edges that have an interior angle greater than 180 degrees (a concave edge), preferably in order of greatest angle.

If the mesh is not closed, a list of border edges should also be kept.

Tracking the Silhouette

A shadow silhouette is an ordered list of silhouette edges that always closes itself (i.e. the last silhouette edge will join up to the first silhouette edge). There may be more than one silhouette for a model depending on its type. A silhouette edge is made up of a triangle face visible to the light and an edge index (0-2) which indicates either the adjacent face is invisible to the light or is non-existent (the edge is a border edge).

We can use silhouettes from the previous frame to create new silhouettes for the next frame. Later, a technique will be described that will explain how new silhouettes are detected.

Generally, there are 3 cases to account for when tracking the changes in silhouettes from frame to frame. When the relative position between model and light position is

- More distant, the silhouette will be larger than the previous frame (see figure 1 and 2).
- Less distant, the silhouette will be smaller than the previous frame (see figure 1 and 3).
- Similar distance, but change in position, the silhouette will shrink and grow, or change shape (see figure 1 and 4, the silhouette shrinks on the under side, and grows on the upper side).

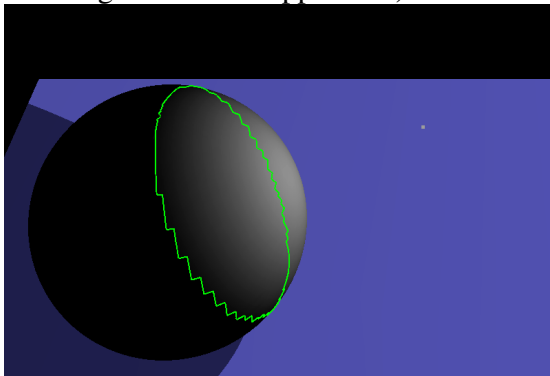


Figure 1 Previous frame

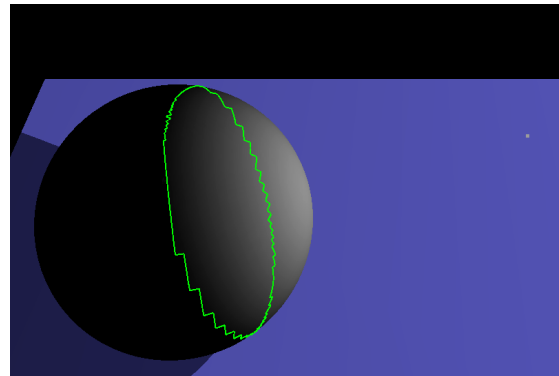


Figure 2 Silhouette becomes larger (encloses more polygon faces)

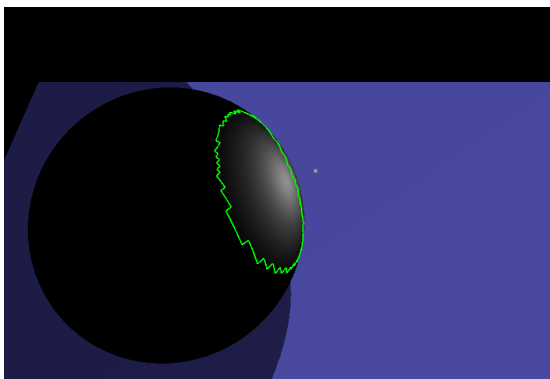


Figure 3 Silhouette becomes smaller (encloses less polygon faces).

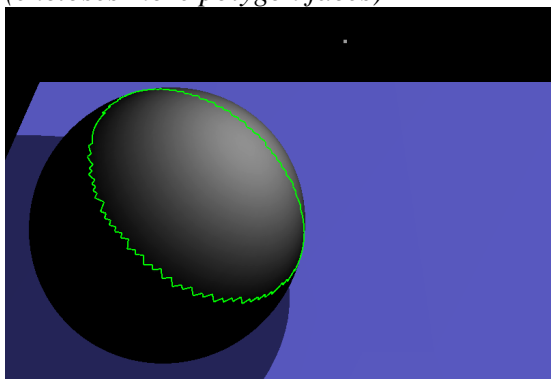


Figure 4 Silhouette changes its shape

In all these cases, a simple method will find a silhouette edge from a previous silhouette edge. By picking an edge on the previous silhouette, check for visibility for adjacent faces F1 and F2 if

- F1 visible XOR F2 visible*, this edge is part of the new silhouette
- F1 visible AND F2 visible*, the silhouette has grown, look for silhouette edge further away from the light position
- F1 is not visible AND F2 is not visible*, the silhouette has shrunk, look for silhouette edge closer to the light position.

*With respect to the light position.

There are various types of algorithms that can be used to choose the path from one edge to another. The direction only has to be general since silhouettes edges never exist in isolation (they are always chained). A silhouette edge will always be found eventually, provided the algorithm takes steps not to go around in circles.

The demo (found at http://www.geocities.com/tom_j_hall) uses a left-right approach to walk the model faces to find the silhouette edge. A cap on iterations prevents infinite loops when the algorithm walks around in circles. Refer to the following procedure for one of the simplest and more efficient methods. For simplicity, this algorithm only looks for the first invisible face from a visible edge, but it is easy to tailor this to find the first visible edge from an invisible edge.

Algorithm Find Silhouette Edge

```

PARAMETERS (Face FromFace, unsigned int EdgeIndex)
LET CurrentFace be FromFace
LET back edge BackEdge be EdgeIndex
LET StepLeftToggle be true
WHILE TRUE
    //check adjacent edges to see if they are silhouettes
    LET LeftFace be adjacent to CurrentFace on edge (BackEdge+1)%3
    LET RightFace be adjacent to CurrentFace on edge (BackEdge+2)%3

    IF LeftFace is non existent or not visible to the light
        RETURN silhouette edge(CurrentFace, (BackEdge+1)%3)
    ELSE IF RightFace is non existent or not visible to the light
        RETURN silhouette edge(CurrentFace, (BackEdge+2)%3)
    ELSE
        IF StepLeftToggle
            LET BackEdge be edge of LeftFace adjacent to
                CurrentFace
            LET CurrentFace be LeftFace
        ELSE
            LET BackEdge be edge of RightFace adjacent to
                CurrentFace
            LET CurrentFace be RightFace
        END IF

        LET StepLeftToggle be NOT StepLeftToggle
    END IF
END WHILE

```

Since all silhouettes are closed loops, once a single edge to a silhouette is found, all edges to the silhouette (with respect to light position P) can be found by using the following algorithm:

Algorithm Complete Silhouette

```
PARAMETERS (Face VisibleFace, unsigned int EdgeIndex)
Append Edge(VisibleFace, EdgeIndex) to the silhouette list
LET CurrentFace be VisibleFace
LET BackEdge be EdgeIndex
WHILE TRUE
  LET TestEdge be (BackEdge+2)%3
  LET TestFace be adjacent to CurrentFace on edge TestFace

  IF TestFace is non existent or is not visible to the light
    IF TestFace equals VisibleFace and TestEdge equals EdgeIndex
      //test to see if we have completed the
      //silhouette loop
      BREAK
    ELSE
      //we have found a silhouette edge
      Append Edge(TestFace ,TestEdge) to the silhouette list
      LET BackEdge be TestEdge
    END IF
  ELSE
    //iterate to the next face
    LET BackEdge be edge of TestFace adjacent to CurrentFace
    LET CurrentFace be TestFace
  END IF
END WHILE
```

In effect, this algorithm hugs the visible faces bordering on the silhouette until it has completed the loop. The resulting silhouette list is ordered, so it is optimized for rendering.

Detecting new silhouettes

In general, models can be separated into two types

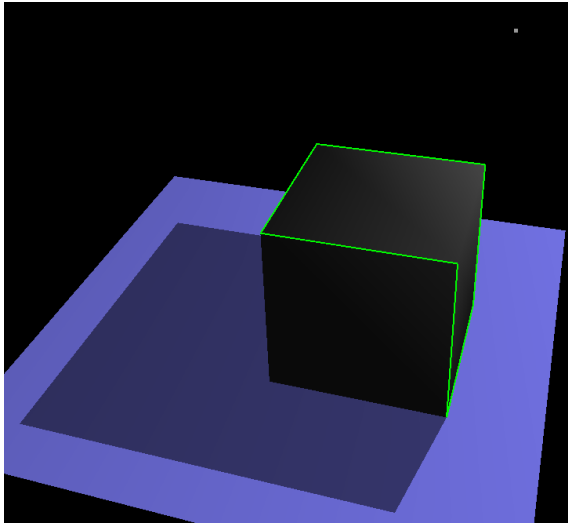
- A *convex* model has all interior angles less than or equal to 180 degrees.
- A *concave* model has 1 or more interior angles found to be greater than 180 degrees.

Similarly, silhouettes may be classified in the same way:

- A *convex* silhouette is made up of no edges that are concave on the model.
- A *concave* silhouette is made up of one or more edges that are concave on the model.

Concave silhouettes are generally more *common* and *easier* to find if we keep track of all concave edges for a model.

Both models types have distinct properties with regard to silhouettes:



Convex models will have no more than 1 *convex* silhouette for any point (see figures 1&5).

Concave models will have no more than $1 + C$ silhouettes for any point where C is the total number of concave edges (see figures 6 & 7).

Figure 5 A typical convex model with its single silhouette highlighted in green.

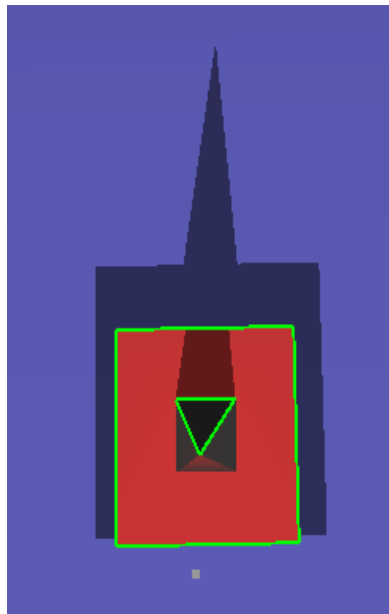
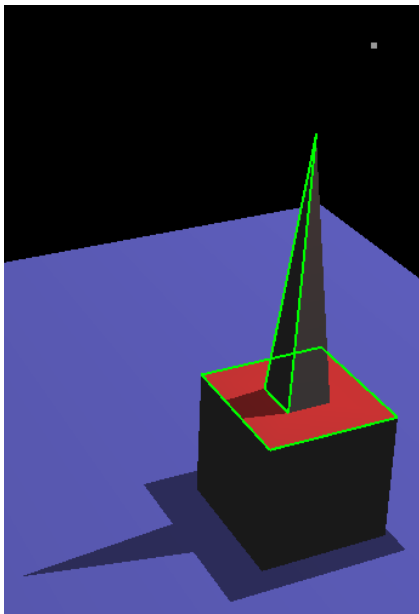


Figure 6 (left) shows a simple concave model with two silhouettes. The model's four concave angles can be found at the base of the spike.

Figure 7 (right) this angle shows more clearly the two distinct silhouettes highlighted in green. The middle silhouette is concave, while the outer silhouette is convex.

A new silhouette can only be formed in two ways, if a concave edge has one face visible and the other face, then an edge has been found (see figures 8,9,10 & 11) and if a non closed mesh has a face that was previously invisible but is now visible (see figures 12 & 13).

Figure 8

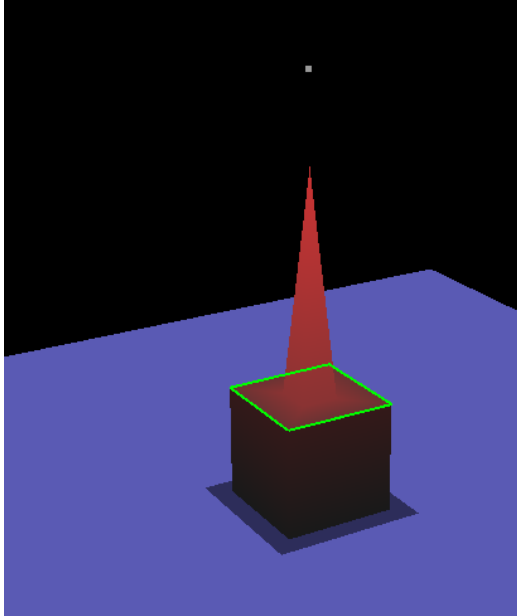


Figure 10

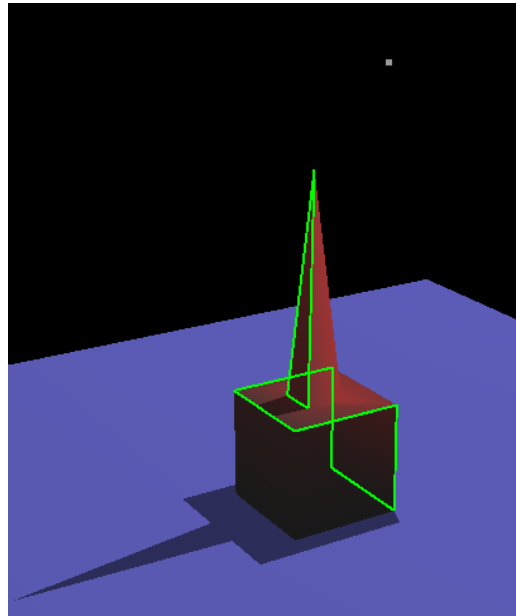


Figure 9

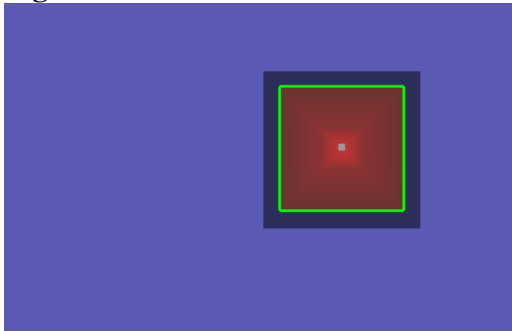


Figure 11

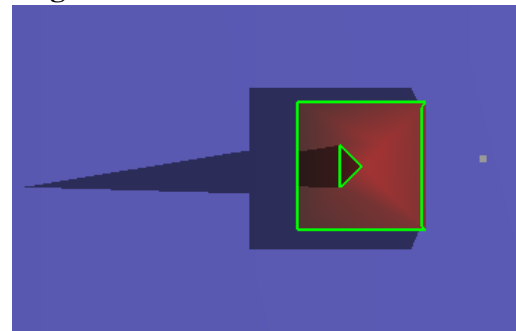


Figure 8 shows a sample concave model with a light (white dot) directly above. **Figure 9** shows a shot from above of the same scene. Only one silhouette (green highlight) is found, but as the light moves to the right, **Figure 10** shows a new silhouette has formed from a concave edge at the base of the spike. **Figure 11** is the same scene from above, and shows more distinctly the two separate silhouettes.

Figure 12

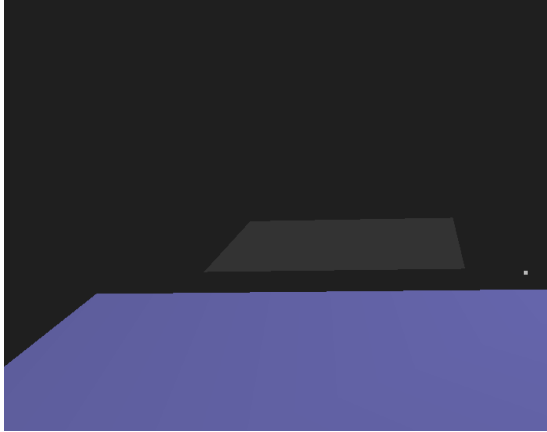


Figure 13

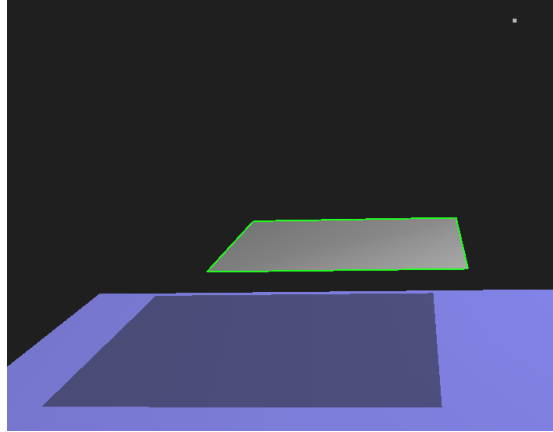


Figure 12 Shows a single plane in space (front facing pointing upwards) that is not visible when the light is below, but as the light is elevated, **Figure 13** shows a newly formed silhouette.

The basic structure for a method that will handle open, or closed models that are concave or convex then, is

```
//Track the changes in existing silhouettes.
FOR each edge E in old silhouette list
    walk from edge E to an edge that is part of the silhouette
    //check if the silhouette edge has already been found
    complete the silhouette
END FOR

//Find new silhouettes in the concave list
FOR each edge E in the concave list
    //check if the edge has been found already
    IF one face of the concave edge is visible and the other invisible
        complete the silhouette
    END IF
END FOR

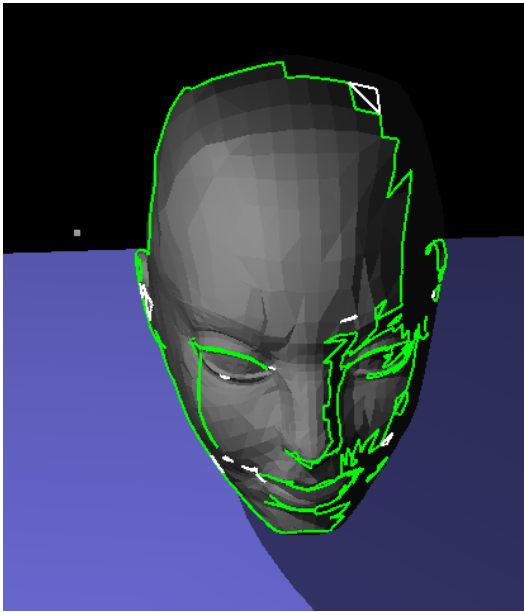
//Finally, look for new silhouettes in the border list
FOR each edge E in the border list
    //check if the edge has been found already
    IF the face is visible
        complete the silhouette
    END IF
END FOR
```

By putting every silhouette edge found into a set, double silhouettes can be avoided by checking if a silhouette edge found already exists in the set.

Further Optimizations

Concavity Level Tuning

Figure 14



A simple but effective optimization is to tune the algorithm to only look for new silhouettes on concave angles that are only greater than a certain degree. For instance, concave angles that are less than 190 degrees generally will not spawn silhouettes. If the concave edge list is ordered from greatest angle to least, this optimization can occur at runtime. If the concavity level is set higher then the algorithm will generally be faster, but at the cost of accuracy. If the tuning is set to its highest (no concave edges are checked) and there are no boundary edges, the algorithm is identical to Markosians randomized silhouette detection algorithm.

Figure 14 shows an example model with its silhouette (with respect to the light) when

only concave angles greater than $(\pi + 1)$ radians are checked for new silhouettes. Silhouettes in white are missed silhouettes. These tend to be small, non boundary silhouettes.

Single edge testing

Since a silhouette loop can be completed from only one silhouette edge, performance would increase further by checking only 1 edge for each existing silhouette.

It is better to use an edge adjacent to a visible face, since it is not possible to stray onto a separate silhouette from the walk if you are 'inside' the silhouette. If a silhouette shrinks entirely, an edge that satisfies this condition does not occur.

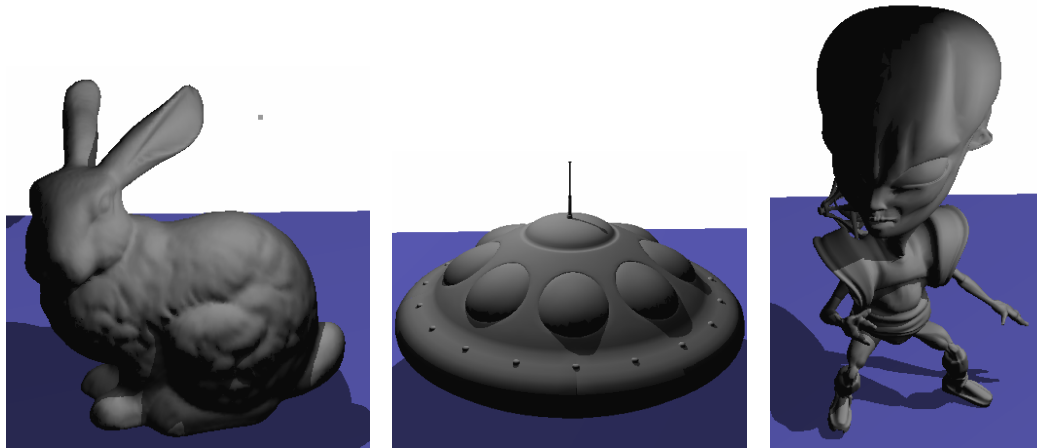
If a record is kept of which old silhouette generates a new silhouette, we can backtrack to the original silhouette and test more edges until a new silhouette is discovered.

This doesn't solve the problem of a silhouette 'splitting' into two silhouettes, one of them a convex silhouette (one that is not made up of any concave angles). This is less likely to occur if the light source is sufficiently far away. This problem could be avoided by tracking the shape of a silhouette, so that the silhouette is checked more thoroughly if the shape changes dramatically from one frame to the next.

Performance

The following statistics were compiled on a Pentium III 650 MHz machine with an NVIDIA TNT2 GPU. The silhouette is extracted from the model with respect to a different position each frame, and only the silhouette is rendered onto the screen. Units are in frames per second.

Test models



From left to right, the famous Stanford bunny (found at the Stanford 3D scanning repository - <http://www-graphics.stanford.edu/data/3Dscanrep>). UFO and Hate Alien found at NVIDIA.

Model	Bunny	UFO	Alien	Speedup Factor ³	Bunny	UFO	Alien
Total Vertices	35947	10016	20667	Tracking 0.0	1.13	1.07	1.04
Total Faces	69451	19608	41265	Tracking 0.01	1.29	1.26	1.10
Brute Force ¹	4.42	18.35	7.73	Tracking 0.1	2.60	2.36	1.57
Tracking ² 0.0	4.98	19.54	8.04	Tracking 0.2	4.28	3.12	2.09
Tracking 0.01	5.68	23.18	8.52	Tracking 0.5	6.90	3.22	2.77
Tracking 0.1	11.47	43.29	12.15	Tracking 1.0	7.88	3.22	3.54
Tracking 0.2	18.92	57.15	16.15				
Tracking 0.5	30.48	59.07	21.44				
Tracking 1.0	34.85	59.15	27.34				

¹ Variation on the exhaustive silhouette detection method. Firstly all front facing polygons are found, then every edge in the front facing polygon set is checked if it is adjacent to a back facing polygon or a border edge.

² Silhouette tracking followed by concavity tuning setting. The number following tracking indicates that only concaves angles greater than $\pi + N$ (N being the number) are checked for new silhouettes.

³ Speedup factor compared to brute force method.

References

[1] Lee Markosian. *Art-based Modeling and Rendering for Computer Graphics*. PhD thesis, Brown University, May 2000.