Adaptive Display Algorithm for Interactive Frame Rates

The Problem
- Two keys for an interactive system
  - Interactive rendering speed: too many polygons – difficult!!
  - Uniform frame rate: varied scene complexity – difficult!!

Possible Solutions
- Visibility Culling – back face culling, frustum culling, occlusion culling (might not be sufficient)
- Levels of Detail (LOD) – hierarchical structures and choose one to satisfy the frame rate requirement

Possible Solutions
- LOD Selections
  - How to pick the optimal ones??!!

Simple Solutions
- The level of detail can be controlled by specifying a threshold
- Static solution: Determine the threshold based on the size or distance of an object to the image plane

Problems
- Do not guarantee uniform frame rate (when the user change his/her view)
- Do not guarantee a bounded frame rate (the visible scene can be arbitrarily complex)

Dynamic algorithms are needed !!!
Dynamic Algorithm

- Reactive – adjust the threshold for LOD based on the feedback from the previous rendering time

- Predictive – estimate the time to render the current scene for all possible levels and to pick the best one (meet the frame rate while maximizing the quality)

Algorithm (funckouser ‘93)

A greedy predictive algorithm

1. Estimate the cost and benefit of each object
2. Maximize the benefit/cost ration

The parameter space

- For each object, we can vary:
  - Level of Detail
  - Rendering Algorithms (flat-shading, Gouraud-shading, Phong-shading etc)

- An object instance: (O,L,R)
  - O: object
  - L: rendered at the level of detail L
  - R: rendering method

Cost and Benefit

- Cost(O,L,R): the time required to render the object
- Benefit (O,L,R): the contribution to the final image
- Goal: Maximize Cost(O,L,R)/Benefit(O,L,R)
- Assumption: CPU is fast enough to keep up

The Challenges

- Find Good Benefit and Cost Estimates
- Real Time Optimization

Cost Estimate (heuristics)

- Estimate the time required to render the object O with LOD L and rendering method R
  (a difficult task - as the rendering time depends on many factors)

Geometry --> Participating Scattering --> Photon Mapping --> Display

rasterization
Cost Heuristics (2)
- Rendering time = max (geometry, rasterization) - assume that geometry and rasterization can be processed in parallel
- Geometry Processing:
  \[ \text{cost}[g] = C_1 \times \# \text{ of polygons} + C_2 \times \# \text{ of vertices} \]
- Rasterization:
  \[ \text{cost}[r] = C_3 \times \# \text{ of pixels} \]

Cost (O,L,R) = Max ( \text{cost}[g] + \text{cost}[r] )

Cost Heuristics (3)
- C1, C2, and C3 needs to be determined empirically - platform and hardware dependent

Benefit Heuristics
- The size of the object
- The accuracy of the object: more detail -> higher quality (compare the image quality)

Accuracy (O,L,R) = 1 - Error
  = 1 - \frac{\text{Base Error}}{\text{Sample}(L,R)^m}

Base Error = 0.5
\text{Sample}(L,R) = \text{number of samples per object}
m = \text{exponent} (\text{Gouraud} = 1, \text{Phong} = 2)

Benefit Heuristics (2)
- Importance: semantics
- Focus: gaze directed
- Motion Blur: slow moving object > fast moving object
- Hysteresis: the difference in LOD between the current and the previous frames (discourage frequent change of LOD)

Benefit Heuristics (3)

Benefit (O,L,R) = \text{Size}(O) \times \text{Accuracy (O,L,R)} \times \text{Importance}(O) \times \text{Focus}(O) \times \text{Motion}(O) \times \text{Hysteresis}(O,L,R)

Optimization Algorithm
- Goal:

Maximize \sum \text{Benefit (O,L,R)}
Subject to:
\sum \text{Cost(O,L,R)} < \text{target frame rate}

This problem is equivalent to the knapsack problem
The Knapsack problem

A thief robbing a store finds $n$ items, the $i$-th item is worth $V_i$ dollars and weighs $W_i$ pounds. The thief wants to take as valuable as possible but he can only carry $W$ pounds in his knapsack (integer knapsack).

Unfortunately, the Knapsack problem is NP-complete.

Greedy Algorithm

Calculate each item’s value per pound ($V_i/W_i$), and the thief takes the items from the highest $V_i/W_i$ until the knapsack is full.

Render the objects with the highest benefit/cost until the time (cost) budget is up.

Results

![Image of a diagram](image1.png)

Results (2)

![Graphs showing performance](image2.png)

Results (3)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Mean_1</th>
<th>Mean_2</th>
<th>Mean_3</th>
<th>Mean_4</th>
<th>Mean_5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Score</td>
<td>0.00</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
</tr>
<tr>
<td>Time</td>
<td>0.43</td>
<td>0.20</td>
<td>0.15</td>
<td>0.18</td>
<td>0.90</td>
</tr>
<tr>
<td>GUI</td>
<td>0.13</td>
<td>0.13</td>
<td>0.13</td>
<td>0.13</td>
<td>0.13</td>
</tr>
</tbody>
</table>

Table 1: Cumulative statistics for the test observer (in seconds).

Possible Projects

1. Verify Funkhouser’s method (how practical is it?)
2. Come up with a similar method for volume rendering
3. Come up with a different method for polygon rendering.