UCL

Acceleration Techniques

V1.2

Anthony Steed

Based on slides from Celine Loscos (v1.0)

UCL

Goals

- Although processor can now deal with many polygons (millions), the size of the models for application keeps on growing
- Want to introduce techniques to generate different options for rendering a specific object (level of detail)
- Want to assess when to use different representations so that the viewer can't notice them in use

2

UCL

Overview

1. Motivation & Introduction
   - Examples
   - Bottlenecks
   - Simple techniques
2. Level of Detail Control
3. Progressive Meshes

3

1. Motivation & Introduction

- Games need always more polygons, more textures
- Also CPU needs to be shared between different components:
  - Sound
  - Animation
  - Behaviour
  - Illumination
  - Etc.
- You need to reduce the rendering cost to control the real time frame rate (50/60fps for games)

4

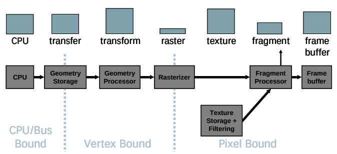
Real time

- You can find in the literature different definitions of real time
  - Often it is assumed 25fps, which comes from videos
  - But if less it is often not noticeable for the eye, and a video running at 15/10 fps seems smooth
  - For games it is 60 fps
  - For some interactive devices with feedback, you need often a frequency of 600hz (or even more)
  - Real time is something that needs to be defined given the applications and the devices
  - In the UCL-CAVE the frame rate is 45 or 42.5 fps /eye
- *Real time* is something that needs to be defined for each application

5

Bottlenecks

- Recall the GPU lecture: bottlenecks occur for many reasons. Two most common being polygon-limited or pixel-limited
  - Reduce the polygons
  - Simplify the shaders



6

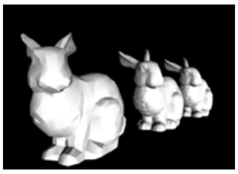
Techniques to accelerate rendering

- Reducing the number of polygons in the model
  - Mesh optimisation
  - Image-based rendering
- Reducing the number of polygons to display
  - Visibility culling
  - Level of detail
  - Image-based rendering
  - Point-based rendering

7

Level of Detail

- Simply edit the mesh to reduce polygon count
  - Some metric of mesh deformation caused by removing edges, faces, etc.
  - Very common as a first step in processing 3D scan data



8

2. Level of Detail Control

- Taken from the article

  Adaptative display algorithm for interactive frame rates during visualisation of complex virtual environments

  Thomas Funkhouser and Carlo Sequin

9

UCL

Context

- Smoothness of the display = constant fps
- Number of polygons to display $\neq$ number of polygons of the model - may vary from one frame to another
- Rendering all potential visible polygons may result in no control on the interactivity

10

UCL

Target

- Control the frame rate: have a constant frame rate whatever needs to be displayed
  - Frame rate decided by the user
- Trade the image quality to achieve the control on the interactive frame rate
  - (Choice often made in practice)
- Idea: select the level of detail and render the visible objects given their importance to achieve the best possible image

11

UCL

Existing techniques considered

- Visibility culling
- Level of detail
- Problem: no guaranty of the bounded frame rate
  - Still more polygons than manageable might need to be displayed
- Reactive vs. predictive
  - It is better to predict the number of polygons that are going to be displayed to pre-adjust the algorithms, rather then being 'caught by surprise' looking at previous frames only

12

UCL

Approach

- Predictive
- Consider 3 parameters
  - object *O*
  - level of detail *L*
  - rendering algorithm (lighting) *R*
- And 2 heuristics
  - *Cost (O,L,R)* : time required to render *O* at *L* with *R*
  - *Benefit(O,L,R)* : the contribution to model perception of *O*
- Goal
  - Maximize Σ *Benefit(O,L,R)*
  - Control Σ *Cost(O,L,R)* $\leq$ *Target Frame Rate*
- *Do as well as possible in a given amount of time*

13

UCL

Cost heuristic

- Predictive = depends on the number of the current visible polygons
- Maximum of time taken by
  - The per-primitive processing
    - Coordinate transformations, lighting, clipping, etc.
  - The per-pixel processing
    - Rasterization, z-buffering, alpha blending, texture mapping, etc.
- $Cost(O,L,R) = C_1 Poly(O,L) + C_2 Vertex(O,L) + C_3 Pix(O,L)$
- C_1, C_2, C_3 constant dependent to the rendering algorithm and the machine

14

UCL

Benefit heuristic

- Ideal: predict the contribution to human perception
  - Difficult to measure
- Practical metrics:
  - Dependent on the size (number of pixels) occupied by the object on the final image
  - Dependent on the accuracy of the rendering algorithm
  - Dependent on other factors
    - Semantic: importance of the object in the scene
    - Focus: place on the screen
    - Motion blur: speed of the object
    - Hysteresis: change in LOD may reduce the quality

15

UCL

Benefit heuristic - Accuracy

- Estimate:
  - The number of errors decreases with the number of samples
    - More mesh/rays, less error
- *Accuracy(O,L,R)*
  = 1 – Error
  = 1 – BaseError/Samples(L,R)^m
- *Samples(L,R)* = Number of pixels/vertices/polygons
- *m* dependent on method (1 = flat, 2 = gouraud)

16

UCL

Benefit heuristic - formula

- Benefit(O,L,R) = Size(O) * Accuracy(O,L,R) * Importance(O) * Focus(O) * Motion(O) * Hysteresis(O,L,R)
- Every function between 0…1

17

UCL

Optimisation algorithm

- Use for each object:
  - Value(O) = Benefit(O,L,R)/Cost(O,L,R)
- Incremental algorithm
  - List all the visible objects
  - Initialise every object
    - visible at previous frame with previous L and R
    - Newly visible with lowest L and R
  - Update accuracy attributes depending on current value
- Loop until stable and under frame rate

18

UCL

Remarks

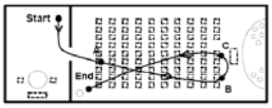
- Worst case: n log n
- But coherence between frame = few iterations
- Parallelisation of the computations/display

19

UCL

Results

- Test scene

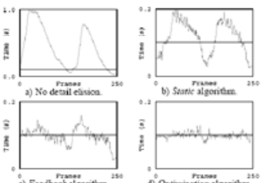


- Results
  - Static: LOD (systematic <1024 pixels)
  - Feedback: LOD with adaptive size threshold
  - Optimization: with prediction

20

UCL

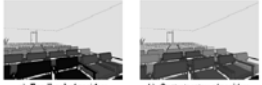
Results



a) No detail elision. b) *Static* algorithm. c) *Feedback* algorithm. d) *Optimization* algorithm.

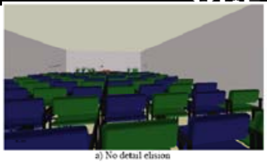
LOD Selection Algorithm	Compute Time		Frame Time		
	Mean	Max	Mean	Max	StdDev
None	0.00	0.00	0.43	0.99	0.305
Static	0.00	0.01	0.11	0.20	0.048
Feedback	0.00	0.01	0.10	0.16	0.026
Optimization	0.01	0.03	0.10	0.13	0.008

21

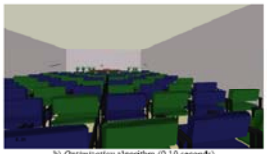


a) *Feedback* algorithm b) *Optimization* algorithm

Figure 9: Images depicting the LODs selected for each object at the observer viewpoints marked 'C' using the *Feedback* and *Optimization* algorithms. Darker shades of gray represent higher LODs.



a) No detail elision



b) *Optimization* algorithm (0.10 seconds)

Figure 10: Images for observer viewpoint 'A' generated using a) no detail elision (72,570 polygons), and b) the *Optimization* algorithm with a 0.10 second target frame time (5,300 polygons).

UCL

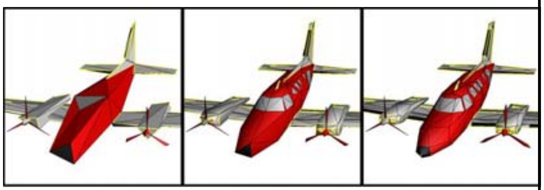
3. Progressive Meshes

- Some objects have a very high polygon-count
- The fine details of the object description are not always needed
- Idea:
  - Lower the number of polygons of an object by reducing its mesh
  - Represent the object with a different polygon count depending on circumstances
    - Level of Detail (LOD)

23

UCL

Example



24

UCL

Applications

- Complex meshes are expensive to store, transmit and render, thus motivating a number of practical problems:
  - Mesh simplification
  - Level-of-Detail (LOD) approximation
  - Progressive transmission
  - Mesh compression
  - Selective refinement

25

UCL

Mesh simplification

- E.g., for scanned data

Scanned model
2 millions polygons

Mesh simplification
7500 polygons

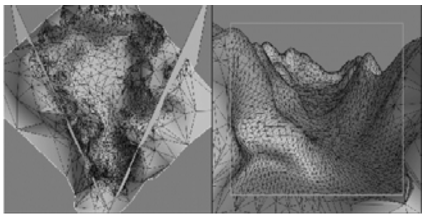
Real Statue

26

UCL

Selective Refinement

- Add detail to specific areas



27

UCL

Progressive meshes

- They are many techniques to calculate and store LOD meshes
- One is
  - Progressive Meshes, H. Hoppe, SIGGRAPH'96

28

UCL

Progressive meshes

- Store a representation of a mesh at different LODs
- Use a structure that makes it easy to go from one level to another
  - Smooth transition is important

29

UCL

LOD structure

- A mesh (made of triangles) can be represented by
  - M(K, V, D, S)
- K = Simplicial complex Connectivity between Mesh elements (faces, edges, vertices)
- V = Vertex positions, define shape of the mesh
- S = Scalar attributes associated to corners {f,v}: colour, normals, texture coordinates
- D = Discrete attributes associated to faces f {j,k,l}

30

Remarks

- The structure is capable of identifying differences within the same object
  - Sharp edges
    - Boundary edge
    - Adjacent faces have different discrete attributes
    - Adjacent corners have different scalar attributes

31

Creation of the progressive mesh

- A representation scheme for storing and transmitting arbitrary triangle meshes

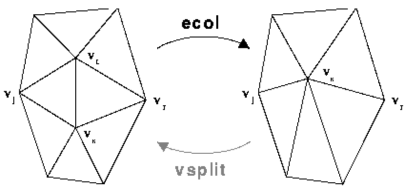
$$\hat{M} = M^n = (M_0, \{ vsplit_0, \ldots, vsplit_{n-1} \})$$

- Mn = Mesh at the higher level
- M0 = Mesh at the lowest level
- vsplit = Vertex split operations between different levels

32

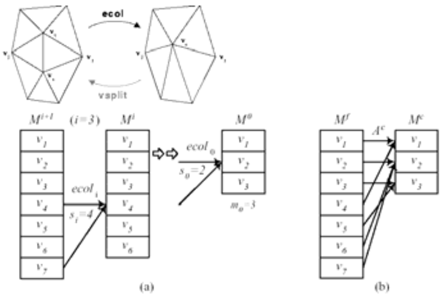
Building the progressive mesh

- Edge collapse / Vertex split



33

Edge collapse



34

Overview of the simplification algorithm

- Energy metric:

$$E(M) = E_{dist}(M) + E_{spring}(M) + E_{scalar}(M) + E_{disc}(M)$$

- Only collapse transformations
- Steps:
  - Priority queue of edge collapse
  - In each iteration perform the transformation at the front of the queue
  - Recompute priorities in the neighbourhood of the transformation

35

Preserving surface geometry

- Record the geometry of the original mesh by sampling from it a set of points
- Evaluating $E_{dist}(V)$ involves computing the distance of each point x_i to the mesh (minimization problem)
- Minimization of $E_{dist}(V) + E_{spring}(V)$ is computed iteratively by:
  - For every V compute optimal parametrizations B
  - For every parametrization B compute optimal vertex position V

36

Preserving Scalar Attributes

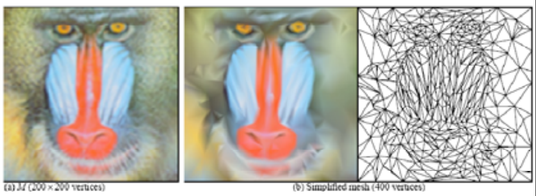
- Continuous scalar fields
- Optimizing scalar attributes at vertices:
  - Each vertex of the original mesh has a position and a scalar attribute v_j
  - We want to measure the deviation of the sampled attribute values X from those of M
  - We introduce a scalar energy term E_{scalar}
  - Solve E_{scalar} by single least-square problems

37

Preserving Scalar Attributes

- Optimizing scalar attributes at corners
  - Partition the corners around a vertex into continuous sets, and solves each continuous set independently for its optimal attribute value.
- Range constraints
- Normals

38



(a) M (200 × 200 vertices) (b) Simplified mesh (400 vertices)

39

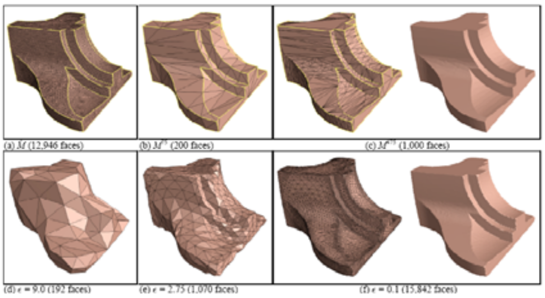
UCL

Preserving discontinuity curves

- Appearance attributes give rise to a set of discontinuity curves in the mesh
- If an edge collapse would modify the topology of discontinuity curves we can either disallow it or penalize it
- We sample an additional set of points X_{disc} from the sharp edges of the original mesh.
- Changes to the topology of the discontinuity curves are allowed, but penalizing those changes.

40

UCL



(a) $\hat{M}$ (12,946 faces) (b) M^{75} (200 faces) (c) M^{475} (1,000 faces)

(d) ϵ = 9.0 (192 faces) (e) ϵ = 2.75 (1,070 faces) (f) ϵ = 0.1 (15,842 faces)

41

UCL

Geomorphs

- Transition between two Meshes M^i and M^{i+1}
  - Popping can occur
  - Can be prevented by using a geomorph M^G
- $M^G(\alpha) = (K^{i+1}, V^G(\alpha))$, $0 \le \alpha \le 1$
  $M^G(0) = M^f$, $M^G(1) = M^c$
- Connectivity of M^{i+1} and vertices linearily interpolated

$$\mathbf{v}_j^G(\alpha) = \begin{cases} (\alpha)\mathbf{v}_j^{i+1} + (1-\alpha)\mathbf{v}_{s_i}^i & , j \in \{s_i, m_0+i+1\} \\ \mathbf{v}_j^{i+1} = \mathbf{v}_j^i & , j \notin \{s_i, m_0+i+1\} \end{cases}$$

42

UCL

Conclusion

- To accelerate the rendering one can reduce the number of polygons to display
- Generic optimisation algorithm to control the frame rate while providing the ‘best possible quality’ based on perception metrics
- Progressive meshes are one example of many different efficient structures to store and retrieve level of detail meshes

43