Professor Jarek
cs7491

Class Notes 2006-1-10

Course Contact Information
www.gvu.gatech.edu/~jarek/courses/7491/
jarek@cc.gatech.edu

On Complexity

Complexity means trouble. Why?

- complexity can make algorithms impracticably slow
- things that are complex cannot easily be rendered in realtime
- transmission delays (transmitting complex things requires lots of bandwidth)
- paging/caching issues (complex things require lots of memory)
- visual clutter (complex scenes and diagrams are difficult to understand)
- mathematically challenging (complex things make our heads hurt)
- complexity leads to unreliable software (complex code is bug-prone)

Our goal is to reduce the complexity of the things we work with.

Broad Principles for Fighting Complexity

- remove redundant information
  - example: backface culling

- Replace or approximate complex components with simple ones
  - example: level of detail (replacing a circle with a decagon)

- change the paradigm
  - example: Image Based Rendering (IBR) instead of polygonal based

- precompute complex things
  - examples: complex animations, occlusion (visibility) testing

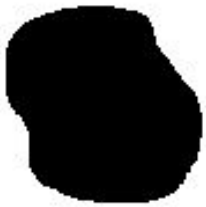
- conservative tests
  - example: using bounding boxes/spheres for collision detection

Complexity Types

- Algebraic (degree of polynomial)
  - example: high-order polynomial surfaces (cones, toruses)
  - solution: replace a torus with a polyhedron

- Topological (see genus, non-manifold)
  - genus is the number of "handles" of a mesh
  - example: a torus (genus 1) is topologically more complex than a sphere (genus 0)

- Morphological (shape features)
  - smoothness or sharpness of a shape
  - how "tricky" the shape is
  - irregular shapes can be considered morphologically complex



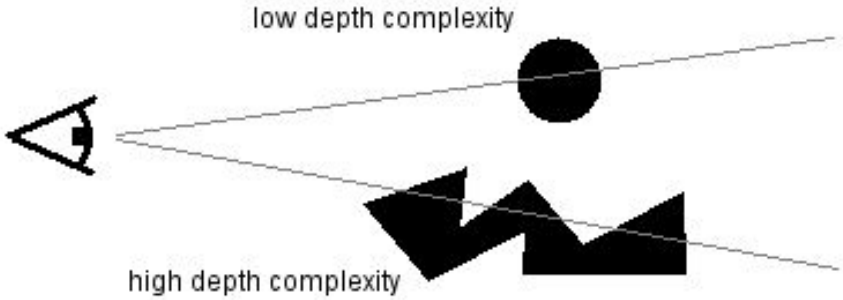
morphologically simple

morphologically complex

- Combinatorial
  - example: triangle count of a mesh

- Representational
  - representing complex data is difficult, relates to visual clutter

- Visual
  - example: depth complexity
  - shapes that are convex have no depth complexity
  - shapes that have many concave parts have high depth complexity



- Dynamic (animation)
  - example: objects (such as walls) buckling under pressure cannot be calculated in real time

Project

See the project web page at http://www.gvu.gatech.edu/~jarek/courses/7491/P1/

Strategy Phase 1

Quantization strategy

- convert double/floating point numbers into integers (12 bits)
- this is a lossy compression (precision is lost)

Replace strategy (linear prediction)

- replace (x_i, y_i) with $(x_i - x_{i-1}, y_i - y_{i-1})$
- this is not lossy, but it doesn't necessarily get us much

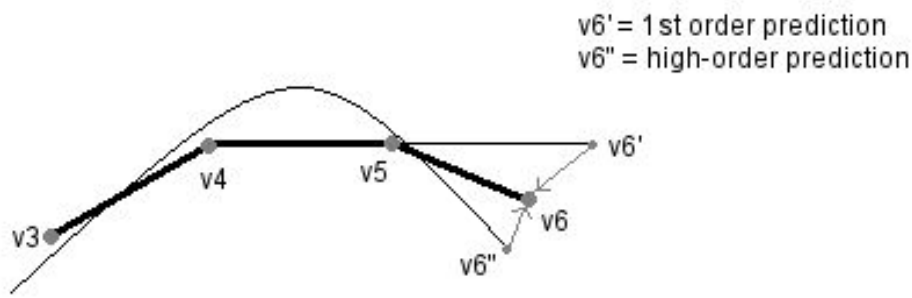
Strategy Phase 2

Prediction (first order)

- replace (x_i, y_i) with $(x_i - (x_{i-1} + (x_{i-1} - x_{i-2})), y_i - (y_{i-1} + (y_{i-1} - y_{i-2})))$
- records the difference between the predicted next point and the actual next point

Prediction (higher order)

- use smoothing techniques, based on the neighbors of points, to more accurately predict the locations of subsequent points
- the differences between the predictions and the actual should be smaller here than using first-order prediction, making the results easier to compress



Strategy Phase 3

If the curve is re-arranged so that all points are equidistant from each other, then only the angle from one point to the next is all that needs to be stored.

More Quantization

Quantization formula:

integer(real x) = round $((x-x_{max}) * (n-1) / (x_{max} - x_{min}))$

where n is the number of bits to an integer. Professor Jarek recommends using

n=12 as a starting point in the project.

Note that when quantizing in many dimensions, compute the unit size (how big, in the original real number base, each integer value is) along the largest dimension. It is recommended that this unit size be consistent for all dimensions.

After quantizing in this manner, the differences between neighboring points will hopefully be very small. Because these differences are integers, high-order bits that are never used can be discarded.