CSE167: Introduction to Computer Graphics

Lecture #12

Jürgen Schulze
University of California, San Diego
Fall 2008
Updates

• Homework project 5 is on-line, due 11/17
  - Topic: a scene graph for RE167
Today

Scene Graphs Part II

• Review

• Rendering scene graphs

Curves

• Introduction

• Polynomial curves
Hierarchical organization
Scene graph for sample scene

TransformGroup

Shape3D
Class hierarchy

TransformGroup

- Stores additional transformation $M$
- Transformation applies to subtree below node
- Monitor-to-world transform $M_0M_1M_2$
Basic rendering

• Traverse the tree recursively

```cpp
TransformGroup::draw(Matrix4 C) {
    C_new = C*M;   // M is a class member for all children
    draw(C_new);
}

Shape3D::draw(Matrix4 C) {
    setModelView(C);
    setMaterial(myMaterial);
    render(myObject);
}
```
Basic rendering

- Traverse the tree recursively

```cpp
TransformGroup::draw(Matrix4 C) {
    C_new = C*M; // M is a class member for all children
    draw(C_new);
}

Shape3D::draw(Matrix4 C) {
    setModelView(C);
    setMaterial(myMaterial);
    render(myObject);
}
```

Initiate rendering with:
```
world->draw(IDENTITY);
```
Performance optimization

- Culling
  - Quickly discard invisible parts of the scene
- Level-of-detail techniques
  - Use lower quality for distant (small) objects
- Scene graph compilation
  - Efficient use of low-level API
  - Avoid state changes in rendering pipeline
  - Render objects with similar properties (geometry, shaders, materials) in batches
Performance optimization

• All serious scene graphs have implementations of these techniques

• Focus on culling today
Level-of-detail techniques

- Don’t draw objects smaller than a threshold
  - Popping artifacts
- Replace objects by impostors
  - Textured planes representing the objects

Dynamic impostor generation

Original vs. impostor
Level-of-detail techniques

• Adapt triangle count to projected size

With bump mapping

Without bump mapping
Culling

- View frustum culling
  - Discard objects outside view frustum
- Occlusion culling
  - Discard objects that are within view frustum, but hidden behind other objects
- Essential for interactive performance with large scenes
Occlusion culling

- Cell-based occlusion culling
  - Divide scene into cells
  - Determine *potentially visible set* (PVS) for each cell
  - Discard all cells not in PVS

- Two main variants
  - Precomputation using binary space partitioning (BSP) trees
  - Portal algorithms

- Specialized algorithms for different types of geometry
  - Indoor scenes
  - Terrain
View frustum culling

- Frustum defined by 6 planes
- Each plane divides space into “outside”, “inside”
- Check each object against each plane
  - Outside, inside, intersecting
- If “outside” all planes
  - Outside the frustum
- If “inside” all planes
  - Inside the frustum
- Else partly inside and partly out
- Efficiency
Bounding volumes

• Simple shape that completely encloses an object
• Generally a box or sphere
• We use spheres
  - Easiest to work with
  - Though hard to get tight fits
• Intersect bounding volume with view frustum, instead of full geometry
Distance to plane

- A plane is described by a point $p$ on the plane and a unit normal $\mathbf{n}$
- Find the (perpendicular) distance from point $x$ to the plane
Distance to plane

- The distance is the length of the projection of $\overrightarrow{x-p}$ onto $\vec{n}$

$$dist = (\overrightarrow{x-p}) \cdot \vec{n}$$
Distance to plane

- The distance has a sign
  - positive on the side of the plane the normal points to
  - negative on the opposite side
  - zero exactly on the plane
- Divides 3D space into two infinite half-spaces

\[ \text{dist}(\mathbf{x}) = (\mathbf{x} - \mathbf{p}) \cdot \mathbf{n} \]
Distance to plane

- Simplification

\[
dist(x) = (x - p) \cdot n \\
= x \cdot n - p \cdot n \\
dist(x) = x \cdot n - d, \quad d = pn
\]

- \(d\) is independent of \(x\)
- \(d\) is distance from the origin to the plane
- We can represent a plane with just \(d\) and \(\hat{n}\)
Frustum with signed planes

- Normal of each plane points outside
  - “outside” means positive distance
  - “inside” means negative distance
Test sphere and plane

- For sphere with radius $r$ and origin $x$, test the distance to the origin, and see if it’s beyond the radius

- Three cases
  - $\text{dist}(x)>r$
    - completely above
  - $\text{dist}(x)<-r$
    - completely below
  - $-r<\text{dist}(x)<r$
    - intersects
Summary

- Precompute the normal $\mathbf{n}$ and value $d$ for each of the six planes.
- Given a sphere with center $\mathbf{x}$ and radius $r$
- For each plane:
  - if $\text{dist}(\mathbf{x}) > r$: sphere is outside! (no need to continue loop)
  - add 1 to count if $\text{dist}(\mathbf{x}) < -r$
- If we made it through the loop, check the count:
  - if the count is 6, the sphere is completely inside
  - otherwise the sphere intersects the frustum
  - *(can use a flag instead of a count)*
Questions?
Culling groups of objects

- Want to be able to cull the whole group quickly
- But if the group is partly in and partly out, want to be able to cull individual objects
Hierarchical bounding volumes

- Given hierarchy of objects
- Bounding volume of each node encloses the bounding volumes of all its children
- Start by testing the outermost bounding volume
  - If it’s entirely out, don’t draw the group at all
  - If it’s entirely in, draw the whole group
Hierarchical culling

• If the bounding volume is partly inside and partly outside
  - Test each child’s bounding volume individually
  - If the child is in, draw it; if it’s out cull it; if it’s partly in and partly out, recurse.
  - If recursion reaches a leaf node, draw it normally
Questions?
Today

Scene Graphs Part II

• Review

• Rendering scene graphs

Curves

• Introduction

• Polynomial curves

• Bézier curves
Modeling

• Creating 3D objects
• How to construct complicated surfaces?

• Goal
  - Specify objects with few control points
  - Resulting object should be visually pleasing (smooth)

• Start with curves, then generalize to surfaces
Usefulness of curves

- Surface of revolution
Usefulness of curves

- Extruded/swept surfaces
Usefulness of curves

• Animation
  - Provide a “track” for objects
  - Use as camera path
Usefulness of curves

- Specify parameter values over time
- 2D curve editor
Usefulness of curves

• Generalize to surface patches
How to represent curves

• Specify every point along a curve?
  - Hard to get precise, smooth results
  - Too much data, too hard to work with

• Specify a curve using a small number of “control points”
  - Known as a *spline curve* or just *spline*
Interpolating splines

- Curve goes through all control points
- Seems most intuitive
- Surprisingly, not usually the best choice
  - Overshoots, wiggles
- Hard to predict behavior
- Hard to get "nice-looking" curves
Approximating splines

- Curve is “influenced” by control points

- Various types & techniques

- Most common: polynomial functions
  - Bézier spline
  - B-spline (generalization of Bézier spline)
  - NURBS (Non Uniform Rational Basis Spline)

- Focus on Bézier splines
**Mathematical definition**

- A vector valued function of one variable \( \mathbf{x}(t) \)
  - Given \( t \), compute a 3D point \( \mathbf{x}=(x,y,z) \)
  - May interpret as three functions \( x(t), y(t), z(t) \)
  - “Moving a point along the curve”
Tangent vector

- Derivative \( x'(t) = \frac{dx}{dt} = (x'(t), y'(t), z'(t)) \)
- A vector that points in the direction of movement
- Length corresponds to speed
Questions?
Today

Curves

- Introduction
- Polynomial curves
- Bézier curves
Polynomial functions

- **Linear:** \( f(t) = at + b \) (1\(^{\text{st}}\) order)
- **Quadratic:** \( f(t) = at^2 + bt + c \) (2\(^{\text{nd}}\) order)
- **Cubic:** \( f(t) = at^3 + bt^2 + ct + d \) (3\(^{\text{rd}}\) order)
Polynomial curves

- Linear
  \[ \mathbf{x}(t) = a t + b \]
  \[ \mathbf{x} = (x, y, z), \ a = (a_x, a_y, a_z), \ b = (b_x, b_y, b_z) \]

- Evaluated as
  \[ x(t) = a_x t + b_x \]
  \[ y(t) = a_y t + b_y \]
  \[ z(t) = a_z t + b_z \]
Polynomial curves

- **Quadratic**: $x(t) = at^2 + bt + c$  
  $(2^{nd} \text{ order})$

- **Cubic**: $x(t) = at^3 + bt^2 + ct + d$  
  $(3^{rd} \text{ order})$

- We usually define the curve for $0 \leq t \leq 1$
Control points

- Polynomial coefficients $a, b, c, d$ can be interpreted as *control points*
  - Remember $a, b, c, d$ have $x, y, z$ components each

- Unfortunately, they don’t intuitively describe shape of curve

- Main objective of curve representation is to come up with intuitive control points
Control points

• How many control points?
  - Two points define a line (1st order)
  - Three points define a quadratic curve (2nd order)
  - Four points define a cubic curve (3rd order)
    - \(k+1\) points define a \(k\)-order curve

• Let’s start with a line...
First order curve

- Based on linear interpolation (LERP)
  - Weighted average between two values
  - “Value” could be a number, vector, color, ...
- Interpolate between points $p_0$ and $p_1$ with parameter $t$
  - Defines a “curve” that is straight (first-order spline)
    - $t=0$ corresponds to $p_0$
    - $t=1$ corresponds to $p_1$
    - $t=0.5$ corresponds to midpoint

$x(t) = Lerp(t, p_0, p_1) = (1-t)p_0 + t p_1$
Linear interpolation

- Three different ways to write it
  - All equivalent
  - Different properties become apparent
1. Weighted sum of the control points
   \[ x(t) = p_0(1 - t) + p_1 t \]
2. Polynomial in \( t \)
   \[ x(t) = (p_1 - p_0)t + p_0 \]
3. Matrix form
   \[ x(t) = \begin{bmatrix} p_0 & p_1 \end{bmatrix} \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} t \\ 1 \end{bmatrix} \]
Weighted average

\[ x(t) = (1 - t)p_0 + \quad (t)p_1 \]

\[ = B_0(t) \ p_0 + B_1(t)p_1, \text{ where } B_0(t) = 1 - t \text{ and } B_1(t) = t \]

- Weights are a function of \( t \)
- Sum is always 1, for any value of \( t \)
- Also known as *blending functions*
Linear polynomial

\[ x(t) = (p_1 - p_0) t + p_0 \]

- Curve is based at point \( p_0 \)
- Add the vector, scaled by \( t \)
Matrix form

\[ x(t) = \begin{bmatrix} p_0 & p_1 \end{bmatrix} \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} t \\ 1 \end{bmatrix} = GBT \]

- Geometry matrix \( G = \begin{bmatrix} p_0 & p_1 \end{bmatrix} \)
- Geometric basis \( B = \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix} \)
- Polynomial basis \( T = \begin{bmatrix} t \\ 1 \end{bmatrix} \)
- In components
  \[ x(t) = \begin{bmatrix} p_{0x} & p_{1x} \\ p_{0y} & p_{1y} \\ p_{0z} & p_{1z} \end{bmatrix} \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} t \\ 1 \end{bmatrix} \]
Tangent

• For a straight line, the tangent is constant

\[ x'(t) = p_1 - p_0 \]

• Weighted average

\[ x'(t) = (-1)p_0 + (+1)p_1 \]

• Polynomial

\[ x'(t) = 0t + (p_1 - p_0) \]

• Matrix form

\[
\begin{bmatrix}
    x'(t)
\end{bmatrix} =
\begin{bmatrix}
    p_0 & p_1
\end{bmatrix}
\begin{bmatrix}
    -1 & 1 \\
    1 & 0
\end{bmatrix}
\begin{bmatrix}
    1 \\
    0
\end{bmatrix}
\]
Questions?
Lissajou curves

http://en.wikipedia.org/wiki/Lissajous_curve
Next time

- Bezier curves
- Curves with multiple segments
- Extending to curves to surfaces