Subdivision, OpenGL Splines

CS116B

Chris Pollett

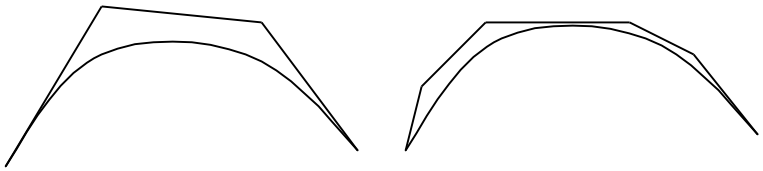
Feb. 21, 2005.

Outline

- Subdivision Methods
- OpenGL Spline Functions

Subdivision and Bezier Curves

- Want to repeatedly split curve sections in half, do enough times, then draw straight lines between very close together sections.
- To subdivide, let **P** be the curve with control points and $\mathbf{p_0}$,$\mathbf{p_1}$, $\mathbf{p_2}$, and $\mathbf{p_3}$. That is, $\mathbf{P}(0)=\mathbf{p_0}$ and $\mathbf{P}(1)=\mathbf{p_1}$. Want to split curve in two parts about $\mathbf{P}(.5)$ into two curves $\mathbf{P_1}$ and $\mathbf{P_2}$.



More Subdivisions

- What are the control points for these two new curves?
- Endpoints have to match curve implies, $\mathbf{p}_{1,0}=\mathbf{p}_0$, $\mathbf{p}_{1,3}=\mathbf{P}(.5)=1/8(\mathbf{p}_0+3\mathbf{p}_1+3\mathbf{p}_2+\mathbf{p}_3)=\mathbf{p}_{2,0}$ and $\mathbf{p}_{2,3}=\mathbf{p}_1$.
- The two curves and must be of equal slope at the meeting point and the first derivative conditions imply: $\mathbf{p}_{1,1}=1/2(\mathbf{p}_0+\mathbf{p}_1)$, $\mathbf{p}_{2,2}=1/2(\mathbf{p}_2+\mathbf{p}_3)$, and $\mathbf{p}_{1,2}=1/4(\mathbf{p}_0+2\mathbf{p}_1+\mathbf{p}_2)$, $\mathbf{p}_{2,1}=1/4(\mathbf{p}_1+2\mathbf{p}_2+\mathbf{p}_3)$
- Note by reusing how we compute these, we can speed up computation of these values.

OpenGL Bezier-Spline Curves

- The general sequence to specify Bezier curve parameters looks like:

  glMapl*(GL_MAP1_VERTEX_3, uMin, uMax, stride, nPts, *ctrlPts);

  glEnable(GL_MAP1_VERTEX_3);

  and can be deactivated with

  glDisable(GL_MAP1_VERTEX_3);

  Here * is f or d. uMin and uMax are the high and low knot values. 0 and 1 for Bezier curves. nPts is the number of control points in the spline. stride -says how to step through the control point array, ctrlPts is the control point array. VERTEX_4 could be used for homogeneous coordinates.

More Open Spline Curves

- To evaluate points between the range we then use glEvalCoord1*(uValue).
- To generate evenly spaced values can use:

  glMapGrid1*(n, u1, u2);

  glEvalMesh1(mode, n1, n2);

  where n controls the number of spaces between u1 and u2. Here mode is GL_POINT or GL_LINE and n1 and n2 give integers that correspond to u1 and u2.

OpenGL Bezier-Spline Surfaces

- To get a surface we use the functions:

  glMap2*(GL_MAP2_VERTEX_3, uMin, uMax, uStride,
  nuPts, vMin, vMax, vStride, nvPts, *ctrlPts);

  glEnable(GL_MAP2_VERTEX_3);

  and to disable

  glDisable(GL_MAP_VERTEX_3);

  To evaluate points use

  glEvalCoord2*(uVal, vVal); or
  glEvalCoord2*v(uvArray);

More OpenGL Bezier Spline Surfaces

- Can generate evenly spaced grids using
  glMapGrid2*(nu, u1, u2, nv, v1, v2);
  glEvalMesh(mode, nu1, nu2, nv1,nv2);

GLU B-Spline Curves

- The basic idea for setting up a B-spline is illustrated by the following code fragment:

```
GLUnurbsObj *curveName;
curveName = gluNewNurbsRenderer();
gluBeginCurve(curveName);
    gluNurbsCurve(curveName, nknots, *knotvector,
    stride, *ctrlpts, degParam, GL_MAP1_VERTEX_3);
glEndCurve(curveName);
```

To delete the curve use:

```
gluDeleteNurbsRenderer(curveName);
```

GLU B-spline Surfaces

- Similar:

```
GLUnurbsObj *surfName;
surfName = gluNewNurbsRenderer();
gluNurbsProperty(surfName, property1, value1);
//GLU_NURBS_MODE, GLU_NURBS_TESSELLATOR
//GLU_DISPLAY_MODE, GLU_OUTLINE_POLYGON
…
gluBeginSurface(surfName);
    gluNurbsSurface(surfName, nuknots, *uknotvector, nvKnots,
    vKnotVector, ustride, vStride, &ctrlpts[0][0][0], uDegParam, vDegParam
    GL_MAP2_VERTEX_3);
glEndSurface(surfName);
```