# Curves and Surfaces in OpenGL

- OpenGL supports Bézier curves and surfaces thourgh mechanisms called **evaluators**.
- These are used to compute values for the Bernstein polynomials of any order.
- Evaluators do not require uniform spacing of control points, and can be used to generate other types of polynomial curves and surfaces.

# Evaluators in OpenGL

- The OpenGL evaluator functions allow you to use a polynomial mapping to produce vertices, normals, texture coordinates, and colors.
- These calculated values are then passed on to the processing pipeline as if they had been directly specified.
- The evaluator functions are also the basis for the NURBS (Non-Uniform Rational B-Spline) functions, which allow you to define curves and surfaces

# Evaluators in OpenGL

- A one-dimensional evaluator is defined by:

```
glMap1f(type,u_min,u_max,stride,order,point_array)
```

- *target* defines the kind of values that are generated by the evaluator, e.g.
  - GL_MAP1_VERTEX_3: Each control point is three floating-point values representing *x, y,* and *z*.
  - GL_MAP1_NORMAL: Each control point is three floating-point values representing the *x, y,* and *z* components of a normal vector.
  - GL_MAP1_TEXTURE_COORD_X: Each control point holds the texture coordinates.

# Evaluators in OpenGL

- *u1, u2*: defines the domain for parameter u.
- *Stride*: The number of floats or doubles between the beginning of one control point and the beginning of the next one in the data structure referenced in *points*. This allows control points to be embedded in arbitrary data structures. The only constraint is that the values for a particular control point must occupy contiguous memory locations.
- *Order*: The number of control points. Must be positive.
- *Points*: A pointer to the array of control points.

# Bézier Curves in OpenGL

- For the one-dimensional evaluator:

```
glMap1f(type,u_min,u_max,stride,order,point_array)
```

- Example:

```
GLfloat pts[4][3] = {{-2.0, -2.0, -1.0},
                     {-1.0, 2.0, 1.0},
                     {1.0, -2.0, -2.0},
                       {2.0, 2.0, 3.0}};
glMap1f(GL_MAP1_VERTEX_3,0.0,1.0,3,4, &pts[0][0]);
glEnable(GL_MAP1_VERTEX_3);   //enable evaluator
```

# Bézier Curves in OpenGL

- Once an evalutor has been set up, we generate the values from the active evaluator as follows with `glEvalCoord1f(u)`:

```
glBegin(GL_LINE_STRIP);
   for (i = 0; i <= NUM_STEPS; i++)
      glEvalCoord1f((GLfloat)i/NUM_STEPS);
glEnd();
```

- Alternatively, if the values of u are equally spaced, we can use:

```
glMapGrid1f(NUM_STEPS,0,1);
glEvalMesh1(GL_LINE,0,NUM_STEPS);
```

# Bézier Surfaces in OpenGL

- Surfaces are generated in a manner similar to curves, using the functions **`glMap2, glEvalCoord2, glMapGrid2f`** and **`glEvalMesh2`** instead.
- For example, set it up with:

```
glMap2f(GL_MAP2_VERTEX_3, 0, 1, 3, 4,
            0, 1, 12, 4, &ctrlpoints[0][0]);
 glEnable(GL_MAP2_VERTEX_3);
```

- then render with:

```
glMapGrid2f(8, 0.0, 1.0, 16, 0.0, 1.0);
glEvalMesh2(GL_LINE,0,NUM_S_STEPS,0,NUM_T_STEPS);
```

# NURBS functions

- Evaluators can be used to generate non-uniform spacing of points also.
- Any polynomial form can be converted to Bezier form by proper generation of control points.
- The OpenGL GLU library simplifies these steps by providing a set of NURBS functions.
- These allow finer control of the shape and rendering of the surface.