```
uniform vec4 min_param, delta_param;
uniform mat4 control_points1, control_points2, control_points3,
control_points4;
varying vec2 parameter;

struct PosNorm
{
    vec4 pos;
    vec4 norm;
};

PosNorm evaluate_surface()
{
    PosNorm pn;
    vec4 weight_d1, weight_d2, weight_d3;
    vec4 weight1, weight2, weight3, weight4;
    vec4 temp1, temp2, temp3;

    temp1 = vec4(1,1,0,0) - gl_Vertex;
    weight_d1 = temp1 * temp1;
    weight_d2 = gl_Vertex * temp1;
    weight_d3 = gl_Vertex * gl_Vertex;
    weight1 = weight_d1 * temp1;
    weight2 = weight_d2 * temp1 * vec4(3,3,0,0);
    weight3 = weight_d3 * temp1 * vec4(3,3,0,0);
    weight4 = weight_d3 * gl_Vertex;
    weight_d2 *= vec4(2,2,0,0);

    temp1 = weight1.x * control_points1[0] + weight2.x * control_points1[1]
          + weight3.x * control_points1[2] + weight4.x * control_points1[3];
    temp2 = weight1.x * control_points2[0] + weight2.x * control_points2[1]
          + weight3.x * control_points2[2] + weight4.x * control_points2[3];
    temp3 = weight1.x * control_points3[0] + weight2.x * control_points3[1]
          + weight3.x * control_points3[2] + weight4.x * control_points3[3];
    pn.norm = weight1.x * control_points4[0] + weight2.x * control_points4[1]
            + weight3.x * control_points4[2] + weight4.x * control_points4[3];
    pn.pos = weight1.y * temp1 + weight2.y * temp2
           + weight3.y * temp3 + weight4.y * pn.norm;

    temp1 = weight_d1.y * temp1 + weight_d2.y * temp2 + weight_d3.y * temp3;
    temp3.w = 1.0 / temp1.w;
    temp1 *= temp3.w;
    temp2 = weight_d1.y * temp2 + weight_d2.y * temp3 + weight_d3.y * pn.norm;
    temp3.w = 1.0 / temp2.w;
    temp2 *= temp3.w;
    pn.norm = temp2 - temp1;

    temp1 = weight1.y * control_points1[0] + weight2.y * control_points2[0]
          + weight3.y * control_points3[0] + weight4.y * control_points4[0];
    temp2 = weight1.y * control_points1[1] + weight2.y * control_points2[1]
          + weight3.y * control_points3[1] + weight4.y * control_points4[1];
    temp3 = weight1.y * control_points1[2] + weight2.y * control_points2[2]
          + weight3.y * control_points3[2] + weight4.y * control_points4[2];
    weight1 = weight1.y * control_points1[3] + weight2.y * control_points2[3]
            + weight3.y * control_points3[3] + weight4.y * control_points4[3];

    temp1 = weight_d1.x * temp1 + weight_d2.x * temp2 + weight_d3.x * temp3;
    temp3.w = 1.0 / temp1.w;
    temp1 *= temp3.w;
    temp2 = weight_d1.x * temp2 + weight_d2.x * temp3 + weight_d3.x * weight1;
    temp3.w = 1.0 / temp2.w;
    temp2 *= temp3.w;
    temp1 = temp2 - temp1;

    pn.norm.xyz = normalize(cross(vec3(temp1), vec3(pn.norm)));

    temp1.w = 1.0 / pn.pos.w;
    pn.pos *= temp1.w;

    parameter = vec2(gl_Vertex * delta_param + min_param);

    return pn;
}
```

Figure 1: Surface evaluation GL shading language function.

```
uniform vec4 param_scale, param_offset;

vec4 evaluate_curve()
{
    vec4 temp1, temp2, temp3;

    temp1 = mix(gl_MultiTexCoord0,gl_MultiTexCoord1,gl_Vertex.x);
    temp2 = mix(gl_MultiTexCoord1,gl_MultiTexCoord2,gl_Vertex.x);
    temp3 = mix(gl_MultiTexCoord2,gl_MultiTexCoord3,gl_Vertex.x);
    temp1 = mix(temp1,temp2,gl_Vertex.x);
    temp2 = mix(temp2,temp3,gl_Vertex.x);
    temp1 = mix(temp1,temp2,gl_Vertex.x);

    temp1 /= temp1.w;
    temp1 = temp1 * param_scale + param_offset;
    temp1 = temp1 * vec4(2,2,0,0) + vec4(-1,-1,1,1);

    return temp1;
}
```

Figure 2: Curve evaluation GL shading language function.

```
uniform sampler2D trimming_texture;
uniform vec4 trimming_enabled;
varying vec2 parameter;

void trimm_surface( )
{
    vec4 threshold;

    threshold = texture2D(trimming_texture, parameter);
    threshold += trimming_enabled - vec4(0.5,0,0,0);
    if(threshold.r < 0) discard;
}
```

Figure 3: Trimming GL shading language function.

Specification of proposed extension

```
Name

    GLU_EXT_nurbs_object

Name Strings

    GLU_EXT_nurbs_object

Contact

    {guthe,edhellon,rk}(at)cs.uni-bonn.de

Status

    XXX - Not complete yet!!!

Version

    Last Modified Date: November 3, 2004
    Author Revision: 0.9

Number

    n.a.

Dependencies

    ARB_shading_language_100 is required.
    ARB_render_texture is required.
    ARB_vertex_buffer_object is required.

    The extension is written against the OpenGL 2.0 Specification.

    Note: The current implementation interferes with
    ARB_shading_language_100.

Overview

    Traditional rendering of trimmed NURBS surfaces requires
    tessellation on the CPU and transmission of the mesh to the
    graphics card.

    This extension provides an interface for rendering trimmed NURBS
    surfaces directly on the GPU. The extension API is designed as
    simple as possible to support future, more efficient hardware
    implementations for higher degree NURBS surfaces.

    An important advantage compare to EXT_nurbs_tessellator is that
    cracks between adjacent patches cannot occur even along trimming
    curves.

IP Status

    No known IP claims.

Issues

  (1) Should other surface types than trimmed NURBS surfaces be
    supported?

    In priciple any surface that can be converted into a piecewise
    Bezier representation can be used (e.g. T-Splines).

  (2) Should the object/screen space error be a property of the
    surface, rather than an argument of the rendering call?

    Probably not, since using a global error for all surfaces is the
    most common scenario.
```

New Procedures and Functions

```
    int gluGenNurbsObjectsEXT(int count);
    void gluDeleteNurbsObjectEXT(int object);

    void gluControlPoints4fvEXT(int object, int usize, int vsize,
                                float *controlpoints);
    void gluKnotVectorUfvEXT(int object, int size, float *knots);
    void gluKnotVectorVfvEXT(int object, int size, float *knots);

    int gluAddTrimmingLoopEXT(int object);
    void gluAddTrimmingCurve3fvEXT(int object, int loop, int size,
                                float *controlpoints, int knotsize,
                                float *knots);
    void gluDeleteTrimmingLoopEXT(int object, int loop);
    void gluDeleteTrimmingEXT(int object);

    void gluDrawNurbsObjectEXT(int object, float error);
    void gluDrawNurbsObjectsEXT(int first, int count, float error);
    void gluDrawNurbsObjectsivEXT(int *objects, int count, float error);

    void gluGetBoundingBoxfvEXT(int object, float *boundingbox);

    void gluNurbsSpecialProgram(int specialprogram);
    const char* gluGetNurbsEvaluateShader();
    const char* gluGetNurbsTrimmingShader();
```

New Tokens

```
    none
```

Additions to Chapter 5.1 of the OpenGL 2.0 Specification (Evaluators)

```
    Direct rendering of trimmed NURBS surfaces is supported using
    NURBS objects. However, these require fragment and vertex programs
    and the p-buffer extension. If these are supported in hardware,
    NURBS objects are a more efficient way of rendering.
    Note, that no callback function can be supplied, since the
    tessellation is performed on the GPU.

int gluGenNurbsObjectsEXT(int count);

    Creates <count> new NURBS objects with consecutive indices. The
    return value is the index of the first NURBS object.

void gluDeleteNurbsObjectEXT(int object);

    Delete a previously generated NURBS object.

void gluControlPoints4fvEXT(int object, int usize, int vsize,
                            float *controlpoints);

    Sets the rational control points of the NURBS surface. The format of
    the control points is (x,y,z,w). Additionally to the pointer to a
    float array, the number of control points in u and v direction has
    to be specified. The size of the array is 4*usize*vsize.

void gluKnotVectorUfvEXT(int object, int size, float *knots);
void gluKnotVectorVfvEXT(int object, int size, float *knots);

    Sets the u- and v-knot vector of the NURBS surface. The number of
    knots has to be given. Note, that the number of knots and the
    according number of control points determine the degree of the
    surface.

int gluAddTrimmingLoopEXT(int object);

    Adds an empty trimming loop to the surface. The return value is the
    index of the generated loop.

void gluAddTrimmingCurve3fvEXT(int object, int loop, int size,
                            float *controlpoints, int knotsize,
                            float *knots);

    Adds a trimming curve to the loop of the specified NURBS object.
    The curves knot vector has to be specified together with the control
    points.

void gluDeleteTrimmingLoopEXT(int object, int loop);

    Deletes the specified trimming loop from the NURBS object.
    Warning: The indices of the consecutive trimming loops of the surface
    are decremented!

void gluDeleteTrimmingEXT(int object);

    Delete all trimming loops of the NURBS object.

void gluDrawNurbsObjectEXT(int object, float error);

    Draws the specified NURBS object. The error either determines the
    absolute approximation error, or, when negative, the relative screen
    space error.

void gluDrawNurbsObjectsEXT(int first, int count, float error);

    Draws <count> consecutive NURBS objects starting with <first>. When
    drawing multiple trimmed surfaces this is faster than rendering each
    of them separately.

void gluDrawNurbsObjectsivEXT(int *objects, int count, float error);

    Draws the NURBS objects specified in the <objects> array. The number
    of objects has to be specified additionally. This function has the
    same performance as gluDrawNurbsObjectsEXT, but is more flexible.

void gluGetBoundingBoxfvEXT(int object, float *boundingbox);

    Retrieve the bounding box of the NURBS object.
    Warning: The bounding box is not guaranteed to be minimal.

void gluNurbsSpecialProgram(int specialprogram);

    Sets the GL shading language program used to tessellate the trimmed
    NURBS surface.
    Note: The function defined by gluGetNurbsEvaluateShader() has
    to be called from the vertex shader to evaluate the surface and the
    function defined by gluGetNurbsTrimmingShader() has to be called
    from the framgent shader to perform the trimming.

const char* gluGetNurbsEvaluateShader();

    Returns a pointer to the evaluation vertex shader function written
    in the GL shading language.
    The defined function is: PosNorm evaluate_surface();

const char* gluGetNurbsTrimmingShader();

    Returns a pointer to the trimming fragment shader function written
    in the GL shading language.
    The defined function is: void perform_trimming();
```

Additions to Chapter 5.4 of the OpenGL 2.0 Specification (Display Lists)

```
    None of the NURBS object functions can be encapsulated in a display
    list.
```

Errors

```
    None so far, but they will be added later.
```

New State

```
    none
```

New Implementation Dependent State

```
    none
```

Sample Code

```
    The example from Chapter 12 of the OpenGL Programming Guide
    (surface.c) using a NURBS object becomes:

    #include <GL/gl.h>
    #include <GL/glu.h>
    #include <GL/glut.h>
    #include <stdlib.h>
    #include <stdio.h>
    GLfloat ctlpoints[4][4][4];
    int showPoints = 0;
    GLuint theNurb;

    void init_surface(void)
    {
        GLfloat knots[8] = {0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0};
        int u, v;
        for (u = 0; u < 4; u++) {
            for (v = 0; v < 4; v++) {
                ctlpoints[u][v][0] = 2.0*((GLfloat)u - 1.5);
                ctlpoints[u][v][1] = 2.0*((GLfloat)v - 1.5);
                if ( (u == 1 || u == 2) && (v == 1 || v == 2))
                    ctlpoints[u][v][2] = 3.0;
                else
                    ctlpoints[u][v][2] = -3.0;
                ctlpoints[u][v][3] = 1.0;
            }
        }
        gluControlPoints4fvEXT(theNurb, 4, 4, &ctlpoints[0][0][0]);
        gluKnotVectorUfvEXT(theNurb, 8, knots);
        gluKnotVectorVfvEXT(theNurb, 8, knots);
    }

    void init(void)
    {
        GLfloat mat_diffuse[] = { 0.7, 0.7, 0.7, 1.0 };
        GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
        GLfloat mat_shininess[] = { 100.0 };
        glClearColor (0.0, 0.0, 0.0, 0.0);
        glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
        glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
        glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
        glEnable(GL_LIGHTING);
        glEnable(GL_LIGHT0);
        glEnable(GL_DEPTH_TEST);
        glEnable(GL_AUTO_NORMAL);
        glEnable(GL_NORMALIZE);
        theNurb = gluGenNurbsObjects(1);
        init_surface();
    }

    void display(void)
    {
        int i, j;
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glPushMatrix();
        glRotatef(330.0, 1.,0.,0.);
        glScalef (0.5, 0.5, 0.5);
        gluDrawNurbsObject(theNurb,-0.5);
        if (showPoints) {
            glPointSize(5.0);
            glDisable(GL_LIGHTING);
            glColor3f(1.0, 1.0, 0.0);
            glBegin(GL_POINTS);
            for (i = 0; i < 4; i++) {
                for (j = 0; j < 4; j++) {
                    glVertex3f(ctlpoints[i][j][0],
                    ctlpoints[i][j][1], ctlpoints[i][j][2]);
                }
            }
            glEnd();
            glEnable(GL_LIGHTING);
        }
        glPopMatrix();
        glFlush();
    }

    void reshape(int w, int h)
    {
        glViewport(0, 0, (GLsizei) w, (GLsizei) h);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluPerspective (45.0, (GLdouble)w/(GLdouble)h, 3.0, 8.0);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        glTranslatef (0.0, 0.0, -5.0);
    }

    void keyboard(unsigned char key, int x, int y)
    {
        switch (key) {
            case `c':
            case `C':
                showPoints = !showPoints;
                glutPostRedisplay();
                break;
            case 27:
                exit(0);
                break;
            default:
                break;
        }
    }

    int main(int argc, char** argv)
    {
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
        glutInitWindowSize (500, 500);
        glutInitWindowPosition (100, 100);
        glutCreateWindow(argv[0]);
        init();
        glutReshapeFunc(reshape);
        glutDisplayFunc(display);
        glutKeyboardFunc (keyboard);
        glutMainLoop();
        return 0;
    }
```

Revision History

```
    First version, 2004/11/03
```