Figure 1: Surface evaluation GL shading language function.

Figure 2: Curve evaluation GL shading language function.

Figure 3: Trimming GL shading language function.

Specification of proposed extension

Name

SGI_TRIM_nurbs_object

Name Strings

SGI_TRIM_nurbs_object

Contact

(pasha, edellos, lk)@ct.cs.uni-bonn.de

Status

ACK - Not complete yet!!!

Version

Last Modified Date; November 3, 2004

Number

1.9

Implements

- Ada_shading_language_1.0 is required.
- Ada_renderer_texture is required.
- Ada_vertex_buffer_object is required.

The extension is written against the OpenGL 2.0 specification.

Notes: The current implementation interferes with
Ada_shading_language_1.0.

Overview

Traditional rendering of trimmed NURBS surfaces requires tessellation in the GPU and transmission of the seed 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 compared to SGI_TRIM_nurbs_tesselator 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 principle any surface that can be converted into a piecewise linear representation can be used (e.g., P-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

- glutDisplayFunc(display);
- glutReshapeFunc(reshape);
- glutInitWindowPosition (100, 100);
- glutInitWindowSize (500, 500);
- glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);

```c
int main(int argc, char** argv)
} 
```

```c
default:
    break;
}
```

```c
void init(void)
```

```c
void init_surface(void)
```

```c
GLuint theNurb;
```

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

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

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

```c
void gluDeleteNurbsObjectEXT(int object);
```

```c
void gluDeleteTrimmingLoopEXT(int object, int loop);
```

```c
float chpoints[4][4][4]; 
```

```c
void gluAddTrimmingLoopEXT(int object);
```

```c
void gluAddTrimmingCurve3fvEXT(int object, int loop, int size, float *controlpoints);
```

```c
void gluDrawNurbsObjectEXT(int object, float error);
```

```c
void gluDrawNurbsObjectivEXT(int *objects, int count, float error);
```

```c
void gluNurbsSpecialProgram(int specialprogram);
```

```c
const char* gluGetNurbsEvaluateShader();
```

```c
void gluGetBoundingBoxfvEXT(int object, float *boundingbox);
```

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

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

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

```c
New Tokens

- none
- New Implementation Dependent State
- sample code

Sample Code

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

```c
#include <stdio.h>
#include <stdlib.h>
#include <GL/glut.h>
#include <GL/gl.h>

(int gluGetNurbsEvaluateShader();

int main(int argc, char** argv)
```

```c
switch (key) 
```

```c
int main(int argc, char** argv)
```