part 8

Martin Samuelčík
http://www.sccg.sk/~samuelcik
Room I4

GLU

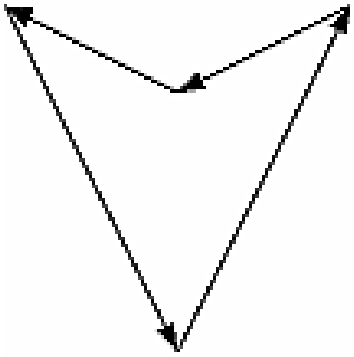
- Library with supporting utility functions for extended work with OpenGL
- Supporting OpenGL functionality
  - Geometry – NURBS, tessellations, quadrics
  - Transformations – projections
  - Textures – scaling, mipmapping
  - Errors handling
- http://www.opengl.org/registry/doc/glu1.3.pdf

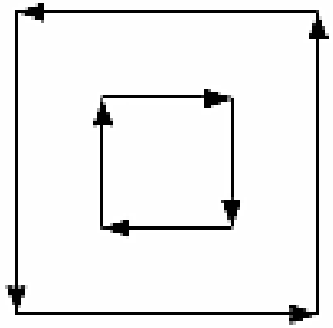
GLU Transformations

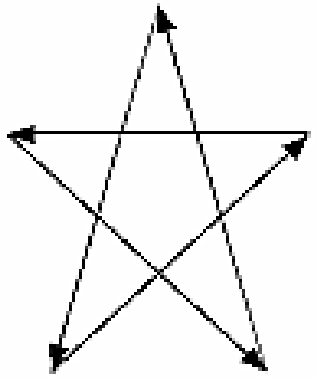
- **int gluUnProject(GLdouble winx, GLdouble winy, GLdouble winz, const GLdouble modelMatrix[16], const GLdouble projMatrix[16], const GLdouble viewport[4], GLdouble *objx, GLdouble *objy, GLdouble *objz)**
  - Transform vertex from window to object coordinates
- **int gluProject(GLdouble objx, GLdouble objy, GLdouble objz, const GLdouble modelMatrix[16], const GLdouble projMatrix[16], const GLdouble viewport[4], GLdouble *winx, GLdouble *winy, GLdouble *winz)**
  - Transform vertex from object to window coordinates

GLU Tessellation

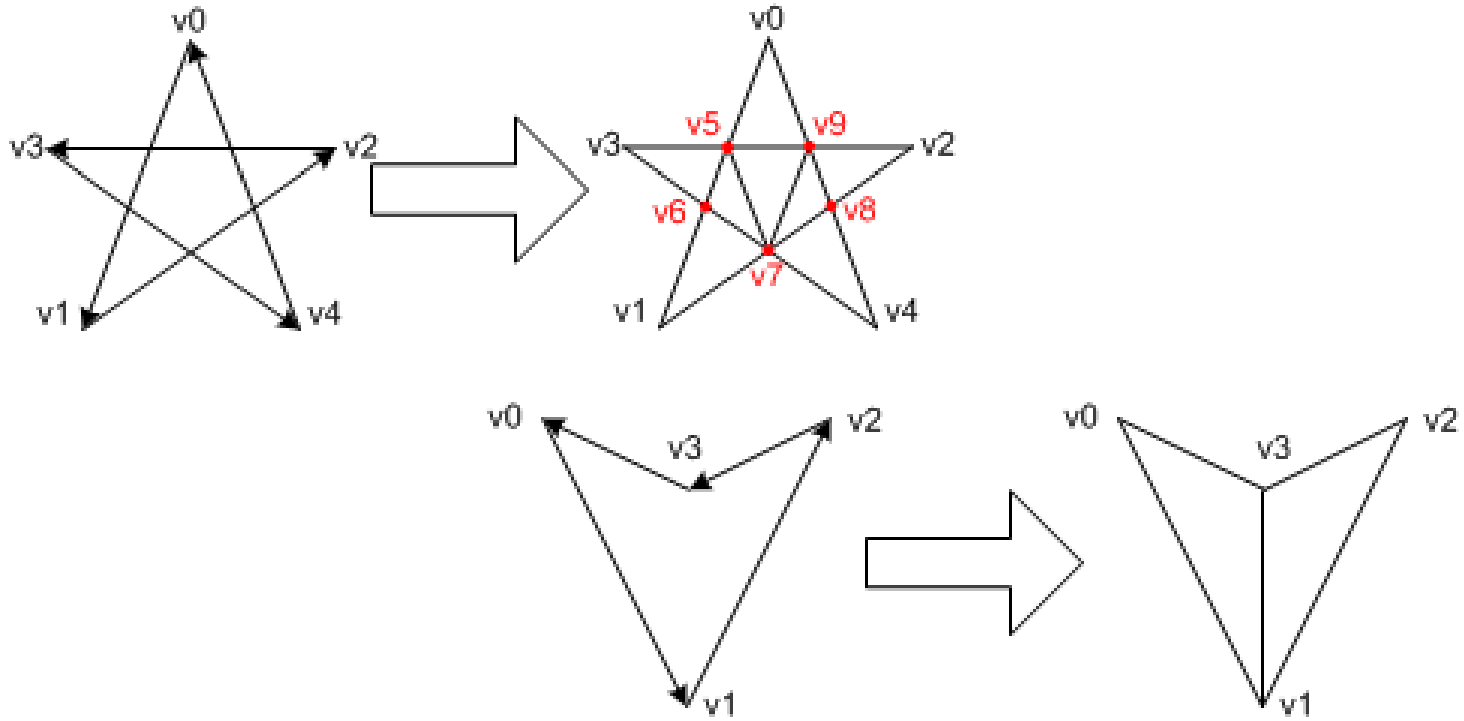
- OpenGL can display only simple convex polygons
- Tessellation – dividing arbitrary polygon into set of simple convex polygons, mainly triangles (triangulation)
- Arbitrary polygons – given by contours







Tessellation



Using GLU tessellation

- Create a new tessellation object
- Register callback functions to perform operations during the tessellation
- Specify tessellation properties
- Create and render tessellated polygons
- Define polygon using set of contours
- Delete tessellation object

Manage tessellation objects

- Tessellation object = type GLUtesselator
- Stores parameters of tesselation and associated data, such as the vertices, edges, and callback functions
- **GLUtesselator* gluNewTess(void);**
  - Creating tessellation object
  - Object can be reused for all tessellations
- **void gluDeleteTess(GLUtesselator *tessobj);**
  - Delete tessellation object

Polygon definition

- **void gluTessBeginPolygon (GLUtesselator *tessobj, void *user_data)**
  - Beginning definition of polygon for tessellation
  - Defined by set of several countors
- **void gluTessEndPolygon (GLUtesselator *tessobj)**
  - Finishing definition of polygon, performing tessellation
- **void gluTessBeginContour (GLUtesselator *tessobj)**
  - Beginning definition of one contour
- **void gluTessVertex (GLUtesselator *tessobj, GLdouble coords[3], void *vertex_data)**
  - Definition of one vertex of contour using coordinates
  - Defined data for vertex, send to vertex and combine callback
- **void gluTessEndContour (GLUtesselator *tessobj)**
  - End of one contour, at this time all vertex data must be present

Example

```
GLdouble rect[4][3] = {50.0, 50.0, 0.0,
                       200.0, 50.0, 0.0,
                       200.0, 200.0, 0.0,
                       50.0, 200.0, 0.0};
GLdouble tri[3][3] = {75.0, 75.0, 0.0,
                      125.0, 175.0, 0.0,
                      175.0, 75.0, 0.0};

GLUtessellator* tobj = gluNewTess();
gluTessBeginPolygon(tobj, NULL);
    gluTessBeginContour(tobj);
      gluTessVertex(tobj, rect[0], rect[0]);
      gluTessVertex(tobj, rect[1], rect[1]);
      gluTessVertex(tobj, rect[2], rect[2]);
      gluTessVertex(tobj, rect[3], rect[3]);
    gluTessEndContour(tobj);
    gluTessBeginContour(tobj);
      gluTessVertex(tobj, tri[0], tri[0]);
      gluTessVertex(tobj, tri[1], tri[1]);
      gluTessVertex(tobj, tri[2], tri[2]);
    gluTessEndContour(tobj);
gluTessEndPolygon(tobj);
```

Tessellation callbacks

- What routines should be called at appropriate times during the tessellation
- **void gluTessCallback(GLUtesselator *tessobj, GLenum type, void (*fn)());**
- To eliminate callback routine, pass null pointer for the appropriate function
- DATA version of callbacks = user defined data are given at the beginning of polygon definition

Callback routines

Type	Callback
GLU_TESS_BEGIN	*void begin (GLenum type);*
GLU_TESS_BEGIN_DATA	*void beginData (GLenum type, void* polygon_data);*
GLU_TESS_EDGE_FLAG	*void edgeFlag (GLboolean flag);*
GLU_TESS_EDGE_FLAG_DATA	*void edgeFlagData (GLboolean flag, void* polygon_data);*
GLU_TESS_VERTEX	*void vertex (void* vertex_data);*
GLU_TESS_VERTEX_DATA	*void vertexData (void* vertex_data, void* polygon_data);*
GLU_TESS_END	*void end (void);*
GLU_TESS_END_DATA	*void endData (void* polygon_data);*
GLU_TESS_COMBINE	*void combine(GLdouble coords[3], void* vertex_data[4], GLfloat weight[4], void** outData);*
GLU_TESS_COMBINE_DATA	*void combineData (GLdouble coords[3], void* vertex_data[4], GLfloat weight[4], void** outData, void* polygon_data);*
GLU_TESS_ERROR	*void error (GLenum errno);*
GLU_TESS_ERROR_DATA	*void errorData (GLenum errno, void* polygon_data);*

Callback routines

- BEGIN – indicating start of primitive, type is GL_LINE_LOOP, GL_TRIANGLE_FAN, GL_TRIANGLE_STRIP, GL_TRIANGLES
- END – indicating end of primitive
- VERTEX – supplying vertex for an primitive, callbacks are called in proper order
- EDGE_FLAG – following vertices are on the boundary or inside polygon, if callback is given, no fans or strips are generated
- COMBINE – invoked for each new created vertex, that is generated as linear combination of 4 vertices using supplied weights
- ERROR – invoked when encountered error

Callbacks example

```
GLUtessellator* tobj = gluNewTess();
gluTessCallback(tobj, GLU_TESS_VERTEX, (GLvoid (*) ()) &vertexCallback);
gluTessCallback(tobj, GLU_TESS_BEGIN, (GLvoid (*) ()) &beginCallback);
gluTessCallback(tobj, GLU_TESS_END, (GLvoid (*) ()) &endCallback);
gluTessCallback(tobj, GLU_TESS_ERROR,  (GLvoid (*) ()) &errorCallback);

// set callbacks for direct rendering of generated triangles
void vertexCallback(double* vertex) {
   glVertex3d(vertex);
}
void beginCallback(GLenum which) {
   glBegin(which);
}
void endCallback(void) {
   glEnd();
}

// callback for writing possible error to console window
void errorCallback(GLenum errorCode) {
   const GLubyte *estring;
   estring = gluErrorString(errorCode);
   fprintf (stderr, "Tessellation Error: %s\n", estring);
   exit (0);
}
```

Callbacks example

```
// all vertices are defined with 6 floats, with 3 coordinates and also with 3 channel color
gluTessCallback(tobj, GLU_TESS_VERTEX, (GLvoid (*) ()) &vertexCallback);
gluTessCallback(tobj, GLU_TESS_COMBINE, (GLvoid (*) ()) &combineCallback);

void vertexCallback(GLvoid *vertex)
{
   const GLdouble *pointer = (GLdouble *) vertex;
   glColor3dv(pointer+3);
   glVertex3dv(vertex);
}

void combineCallback(GLdouble coords[3], GLdouble *vertex_data[4], GLfloat weight[4],
     GLdouble **dataOut )
{
   GLdouble * vertex = (GLdouble *) malloc(6 * sizeof(GLdouble));
   // copy coordinates of new vertex to user defined array for this vertex
   vertex[0] = coords[0]; vertex[1] = coords[1]; vertex[2] = coords[2];
   // compute new color for new vertex as linear combination of colors from 4 old vertices
   for (int i = 3; i < 6; i++)
      vertex[i] = weight[0] * vertex_data[0][i]
                + weight[1] * vertex_data[1][i]
                + weight[2] * vertex_data[2][i]
                + weight[3] * vertex_data[3][i];
   *dataOut = vertex;
}
```

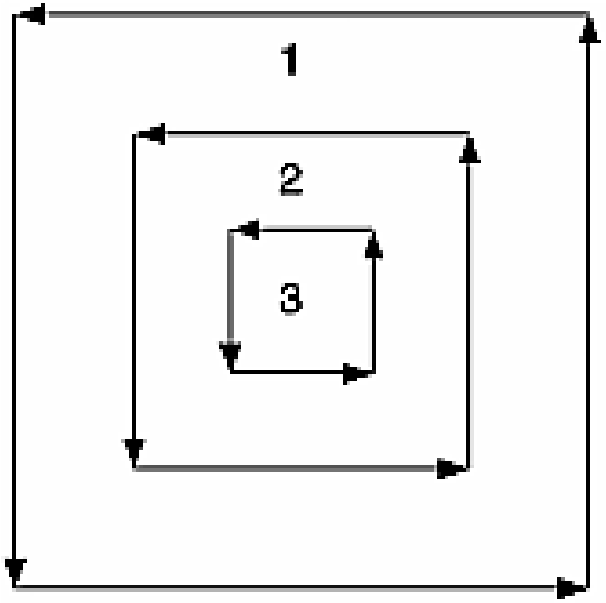
Tessellation properties

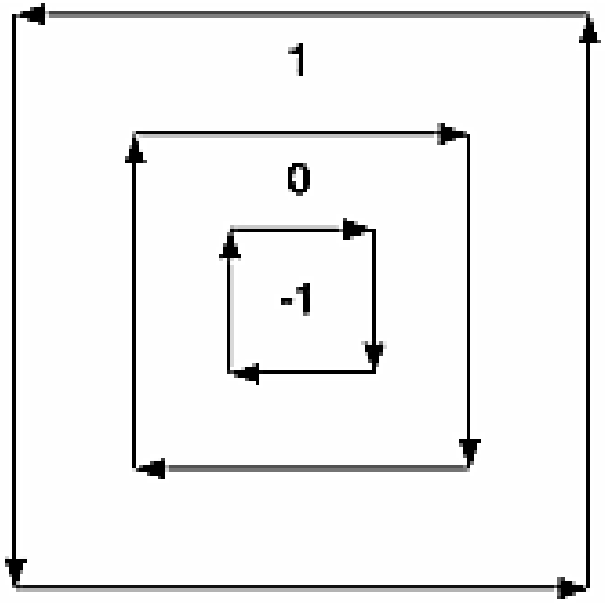
- **gluTessNormal(GLUtesselator *tessobj, GLdouble x, GLdouble y, GLdouble z)**
  - Setting normal of projection plane
  - For normal with null coordinates, automatic determination of projection plane is used
- **void gluTessProperty(GLUtesselator *tessobj, GLenum property, GLdouble value)**
  - property = GLU_TESS_BOUNDARY_ONLY, GLU_TESS_TOLERANCE, GLU_TESS_WINDING_RULE
  - BOUNDARY – if only boundary edges are drawn, only GL-LINE_LOOP primitives are generated
  - TOLERANCE – for merging vertices

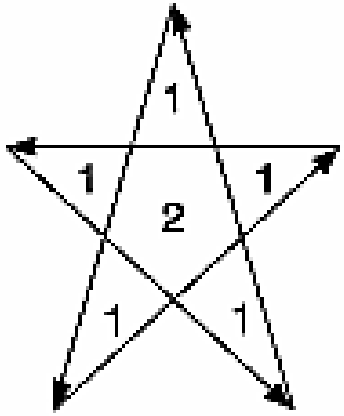
Winding rule

- Winding numbers
  - Numbering of regions defined by contours
  - For a point x and contour C, it is number of revolutions around x if we travel once around C
  - CCW direction adds 1, CW direction subtracts 1
- Determination if region is inside or outside of polygon based on winding number
- Setting GLU_TESS_WINDING_RULE property of tessellation object
  - GLU_TESS_WINDING_ODD (the default), GLU_TESS_WINDING_NONZERO, GLU_TESS_WINDING_POSITIVE, GLU_TESS_WINDING_NEGATIVE, or GLU_TESS_WINDING_ABS_GEQ_TWO

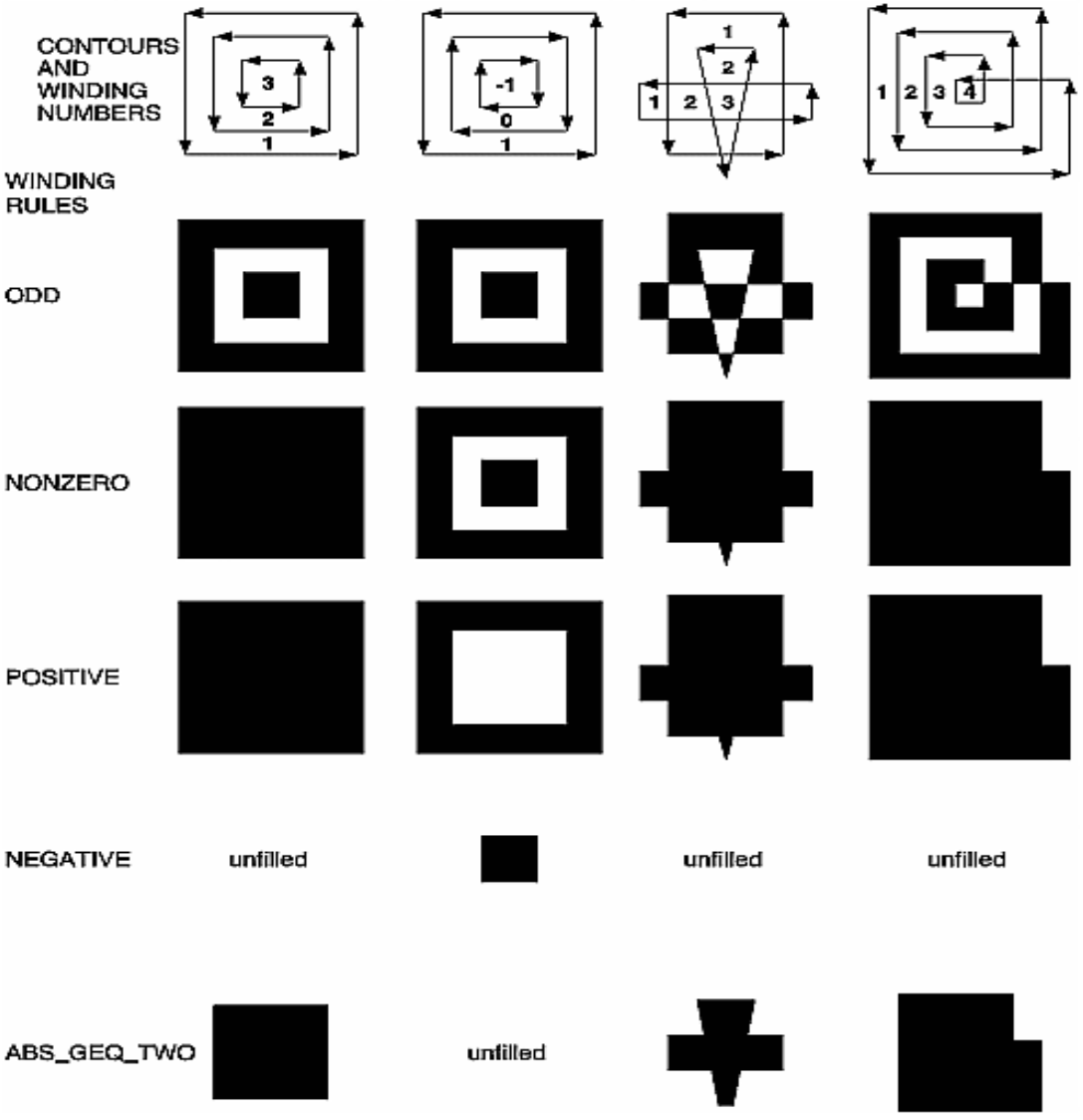
Winding numbers





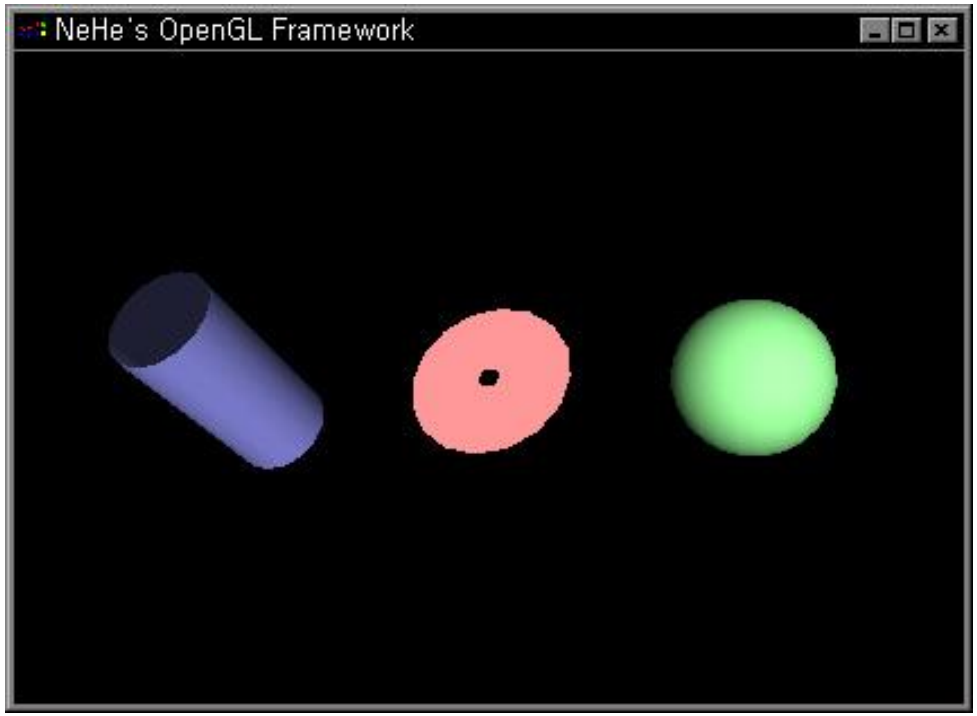


Winding rules



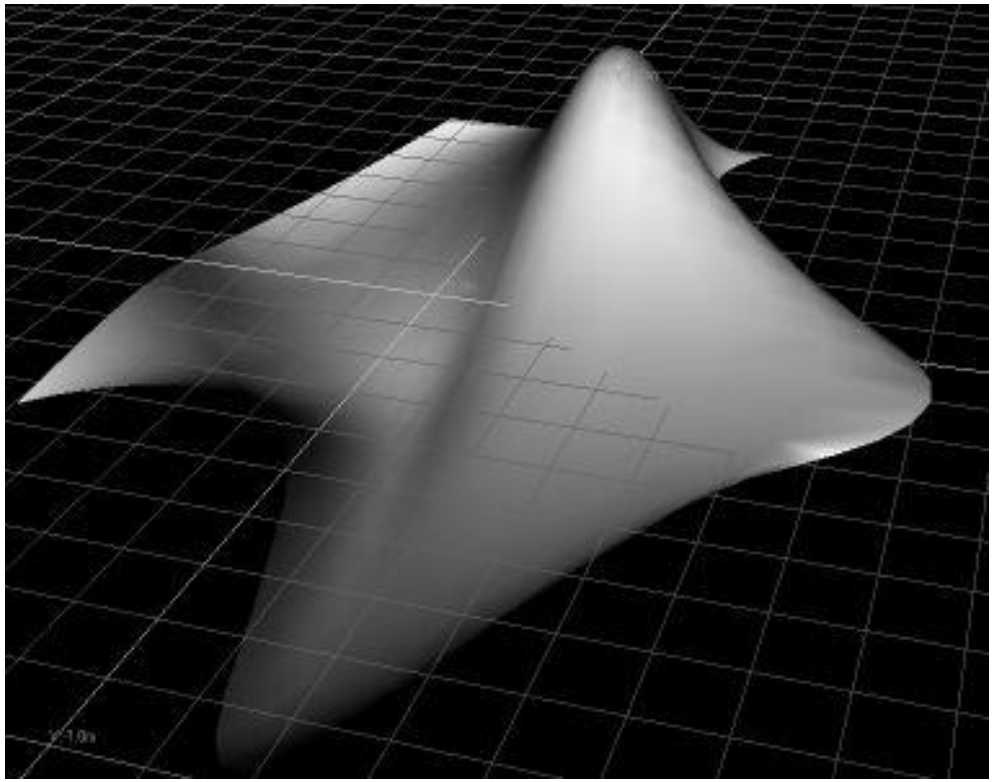
GLU Quadrics

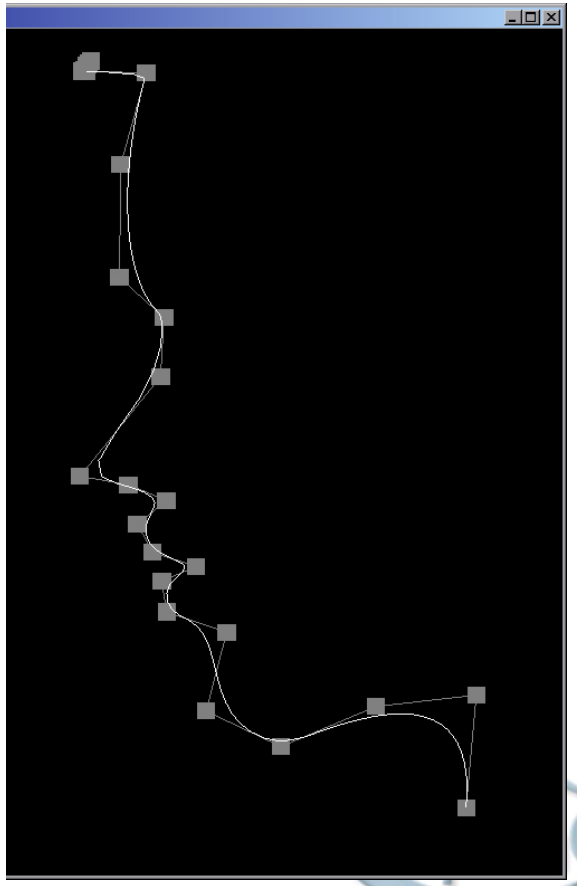
- $a_1x^2 + a_2y^2 + a_3z^2 + a_4xy + a_5yx + a_6xz + a_7x + a_8y + a_9z + a_{10} = 0$
- Sphere, cylinder, disk
- Textured, with normals



GLU NURBS

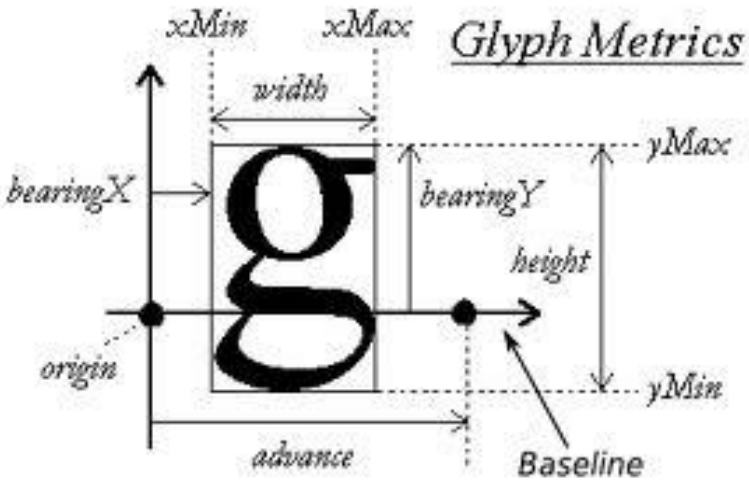
- Representing piecewise rational curves and surfaces – its approximation using triangles and quads

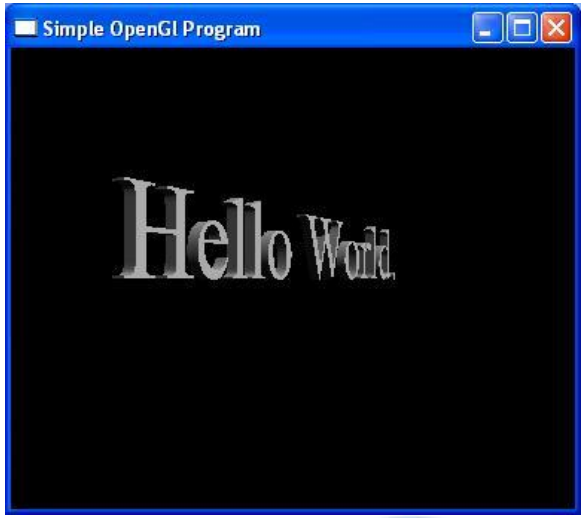




Fonts

- No native support for texts, fonts in OpenGL
- Adding text to virtual scene, each character is defined using template - glyph
- 2D text – bitmap fonts, representing by textures
- 3D text – outline fonts, representing by polygons





GLUT bitmap fonts

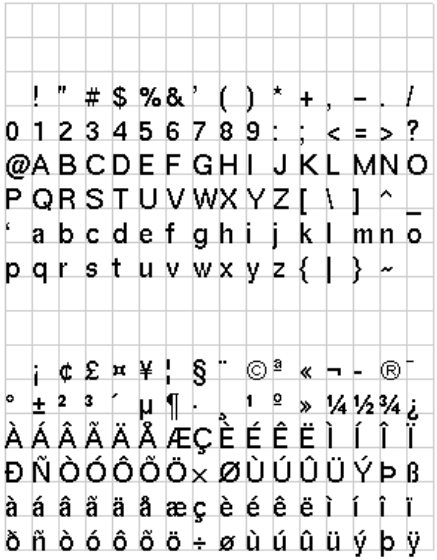
- **void glutBitmapCharacter(void *font, int character)**
  - Rendering one 2D character using one chosen font style
  - Position of character is given by current raster position multiplied by modelview and projection matrices
  - font - GLUT_BITMAP_9_BY_15, GLUT_BITMAP_8_BY_13, GLUT_BITMAP_TIMES_ROMAN_10, GLUT_BITMAP_TIMES_ROMAN_24, GLUT_BITMAP_HELVETICA_10, GLUT_BITMAP_HELVETICA_12, GLUT_BITMAP_HELVETICA_18
- **int glutBitmapWidth(void *font, int character)**
  - Getting width of character, in pixels
- **int glutBitmapLength(void *font, const unsigned char *string)**
  - Getting length of string, in pixels

GLUT bitmap fonts example

```
void renderBitmapString(float x, float y, void *font, const char *string)
{
   glMatrixMode(GL_MODELVIEW);
   glPushMatrix();
   glLoadIdentity();
   glMatrixMode(GL_PROJECTION);
   glPushMatrix();
   glLoadIdentity();
   gluOrtho2D(0, window_width, 0, window_height);
   glRasterPos2f(x, y);
   for (const char* c = string; *c != '\0'; c++)
   {
      glutBitmapCharacter(font, *c);
   }
   glPopMatrix();
   glMatrixMode(GL_MODELVIEW);
   glPopMatrix();
}
```

GLUT_BITMAP_HELVETICA_18

The quick brown
fox jumps over
the lazy dog.

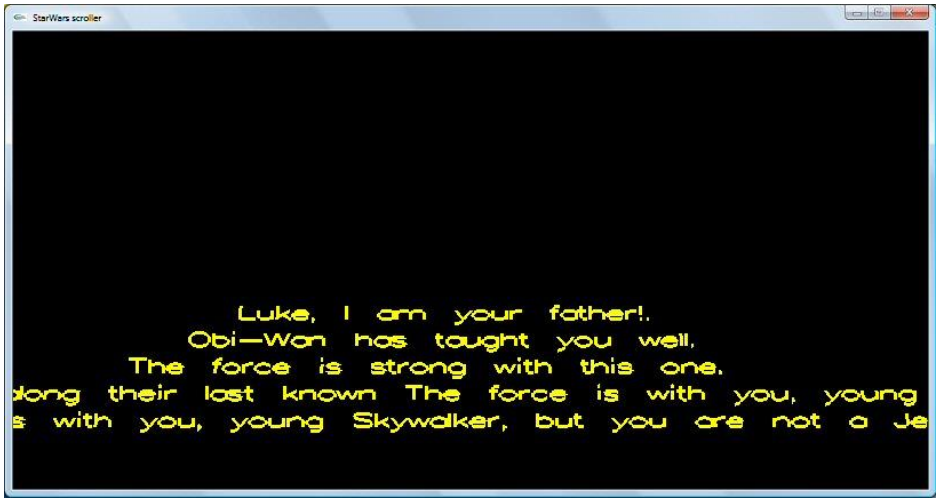


GLUT outline fonts

- **void glutStrokeCharacter(void *font, int character)**
  - Rendering one 3D character using one chosen font style
  - Modelview matrix is used to set position and to advance to next character
  - font - GLUT_STROKE_ROMAN, GLUT_STROKE_MONO_ROMAN
- **int glutStrokeWidth(void *font, int character)**
  - Getting width of character, in modeling units
- **int glutStrokeLength(void *font, const unsigned char *string)**
  - Getting length of string, in modeling units

GLUT stroke fonts example

```
void renderStrokeString(float x, float y, float z, float scale, void *font, const
    char *string)
{
   glMatrixMode(GL_MODELVIEW);
   glPushMatrix();
   glTranslatef(x, y, z);
   glScalef(scale, scale, scale);
   for (const char* c = string; *c != '\0'; c++)
   {
      glutStrokeCharacter(font, *c);
   }
   glPopMatrix();
}
```



WGL fonts

- WGL – supporting library for system dependent OpenGL functionality on Windows platform
- Strong connection with Win32 API
- Creating bitmap and outline fonts from font definitions presented in system
- Each character in created font is represented as display list
- Functions for creating display lists from system fonts

WGL bitmap fonts

- **BOOL WINAPI wglUseFontBitmaps (HDC hdc, DWORD first, DWORD count, DWORD listBase)**
  - Hdc – identifier of device context, whose current font will be used
  - First – first glyph in given font to be used
  - Count – number of glyphs
  - listBase – starting id of cretaed display lists

WGL bitmap fonts example

```
GLuint fontBitmapBase;

GLvoid buildFont(GLvoid)
{
   fontBitmapBase = glGenLists(96);
   HFONT font = CreateFont(-24, 0, 0, 0, FW_BOLD, FALSE, FALSE, FALSE, ANSI_CHARSET, OUT_TT_PRECIS,
      CLIP_DEFAULT_PRECIS,  ANTIALIASED_QUALITY, FF_DONTCARE|DEFAULT_PITCH, "Courier New");
   HFONT oldfont = (HFONT)SelectObject(hDC, font);
   wglUseFontBitmaps(hDC, 32, 96, fontBitmapBase);
   SelectObject(hDC, oldfont);
   DeleteObject(font);
}
GLvoid renderBitmapText(float x, float y, const char *text)
{
   glMatrixMode(GL_MODELVIEW);
   glPushMatrix();
   glLoadIdentity();
   glMatrixMode(GL_PROJECTION);
   glPushMatrix();
   glLoadIdentity();
   gluOrtho2D(0, window_width, 0, window_height);
   glRasterPos2f(x, y);
   glPushAttrib(GL_LIST_BIT);
   glListBase(fontBitmapBase - 32);
   glCallLists(strlen(text), GL_UNSIGNED_BYTE, text);
   glPopAttrib();
   glPopMatrix();
   glMatrixMode(GL_MODELVIEW);
   glPopMatrix();
}
```

WGL outline fonts

- **BOOL WINAPI wglUseFontOutlines(HDC hdc, DWORD first, DWORD count, DWORD listBase, FLOAT deviation, FLOAT extrusion, int format, LPGLYPHMETRICSFLOAT lpgmf)**
  - hdc – identifier of device context, whose current font will be used
  - first – first glyph in given font to be used
  - count – number of glyphs
  - listBase – starting id of cretaed display lists
  - deviation – chordal deviation
  - extrusion – extrusion of font in negative z direction
  - format - WGL_FONT_LINES or WGL_FONT_POLYGONS for displaying only edges or whole filled polygons
  - lpgmf – returned glyph's metric

WGL outline fonts example

```
GLuint fontOutlineBase;
GLYPHMETRICSFLOAT gmf[256];

GLvoid buildFont(GLvoid)
{
   HFONT font, oldfont;
   fontOutlineBase = glGenLists(256);
   HFONT font = CreateFont(-24, 0, 0, 0, FW_BOLD, FALSE, FALSE, FALSE, ANSI_CHARSET, OUT_TT_PRECIS,
      CLIP_DEFAULT_PRECIS,  ANTIALIASED_QUALITY, FF_DONTCARE|DEFAULT_PITCH, "Courier New");
   HFONT oldfont = (HFONT)SelectObject(hDC, font);
   wglUseFontOutlines(hDC, 0, 255, fontOutlineBase, 0.0f, 0.2f, WGL_FONT_POLYGONS,          gmf);
   SelectObject(hDC, oldfont);
   DeleteObject(font);
}

GLvoid renderOutlineText(float x, float y, floatz, float scale, const char *text)
{
   glMatrixMode(GL_MODELVIEW);
   glPushMatrix();
   glTranslatef(x, y, z);
   glScalef(scale, scale, scale);
   glPushAttrib(GL_LIST_BIT);
   glListBase(fontOutlineBase);
   glCallLists(strlen(text), GL_UNSIGNED_BYTE, text);
   glPopAttrib();
   glPopMatrix();
}
```

TrueType fonts

- Glyphs are described using outline curves
- File formats – .ttf, .otf, and many more
- Library for importing and managing TrueType fonts – FreeType
  - http://www.freetype.org/freetype2/
- Library creates textures from outline definition of glyphs
- Using for rendering bitmap fonts
- Adding alpha to texels – antialiasing
- Other libraries – GLTT, OGLFT, FTGL

TrueType fonts

- Loading font from external file
- Storing each glyph in OpenGL intensity texture
- Storing each glyph parameters
- Rendering each character as mapped texture on screen aligned rectangle
- Using intensity from texture as alpha
- Position of rectangle is combined using position of text and glyph parameters

TrueType fonts

The Quick *brown* fox
jumps over **the lazy** dog.
Now is the time for all good men to come to the aid of the party.
Ég get etið gler án þess að meiða mig.
私はガラスを食べられます。それは私を傷つけません。
$\mathbb{N} \subseteq \mathbb{Z} \subset \mathbb{Q} \subset \mathbb{R} \subset \mathbb{C}$

The End!

Questions?