part 9

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

Extensions

- Simple addition of new functionality
- Accommodation of new hardware and software innovations
- Defined by graphics companies or review board – group of experts
- Each extension defined using specification
  - Registry of specifications
  - Almost 500 records, extensions
  - http://www.opengl.org/registry/

Extensions

- Bringing new functions, types, constants, tokens
- Sometimes cooperation with already existing extensions or core functionality
- Extensions syntax
  - Name - XXX_new_extension
  - Tokens - …_XXX
  - Types – GL…XXX
  - Functions – gl…XXX(…)
  - XXX – vendor specific flag – SGI, IBM, HP, WIN, SUN, NV, ATI, EXT, ARB, …
  - Extensions for multiple API – GL_, GLU_, WGL_, GLX_, …

Querying extensions

- Getting list of names of all supported extensions
  - **glGetString(GL_EXTENSIONS);**
  - Use extension only if it is in the list
- Defining all tokens, types and constants – glext.h
- Extension function pointer must be extracted from driver
  - Win platform - **PROC WINAPI wglGetProcAddress(LPCSTR funcName)**
- GLEW library – doing previous work for you

Core specification

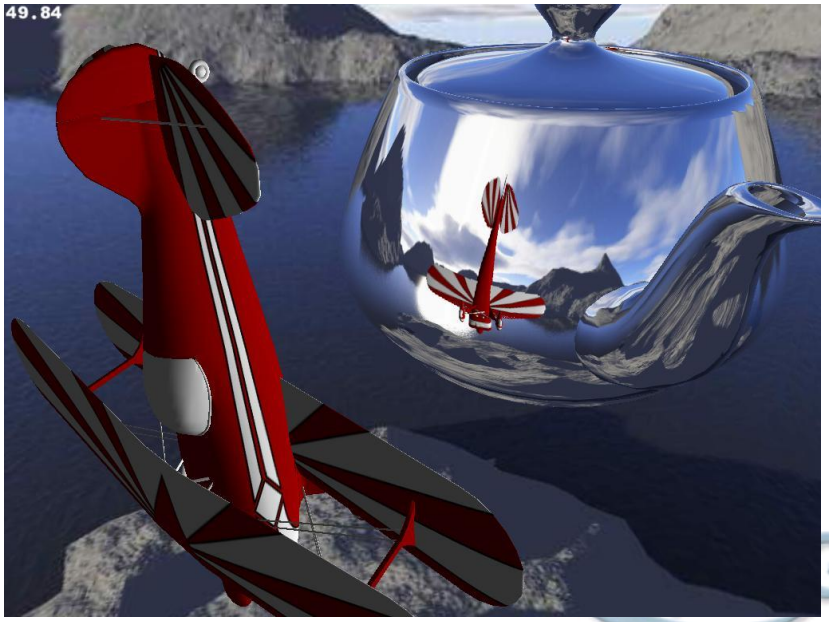
- When some extension proves to be useful and board justified that, it is moved to core specification
  - Adding extensions specification to core specification – tokens, types, functions lost XXX prefix, sometime also change in names
  - Raising number of core specification
  - From first version 1.0 to current version 4.4
  - http://www.opengl.org/documentation/current_version/
  - Sometimes changes to whole specification – depreciated functionality since version 3.2

Querying OpenGL version

- **const GLubyte* glGetString(GLenum name)**
  - GL_VENDOR – company responsible for this GL implementation
  - GL_RENDERER – name of the renderer, hardware platform
  - GL_VERSION – version or release number
  - GL_SHADING_LANGUAGE_VERSION – version or release number for the shading language
- Extension viewers
  - http://www.realtech-vr.com/glview/
  - http://www.ozone3d.net/gpu_caps_viewer/

Texture extensions

- Extending work with textures
- Applying multiple textures on one polygon
- Storing texture data in compressed form
- Using six 2D textures as one cube texture



Multitextures

- Mapping more than one texture on the surface of objects
- For each polygon, we need multiple sets of texture coordinates
- For each polygon, we need multiple current OpenGL textures that will be mapped
- In OpenGL core specification since version 1.3
- Extending existing functionality for OpenGL textures

Multitexturing

- Texture unit
  - Corresponds to one texture, its parameters and its texture coordinates
  - OpenGL tokens for units - GL_TEXTURE0,GL_TEXTURE1,…,GL_TEXTURE0 + i
  - Maximum number of units – **glGetIntegerv (GL_MAX_COMBINED_TEXTURE_IMAGE_UNITS, &texture_units);**
- **void glActiveTexture (GLenum texture), void glActiveTextureARB (GLenum texture)**
  - Setting one texture unit as active
  - Subsequent calls that set parameters of texture are applied to current texture unit

Multitexturing

- **void glMultiTexCoord2f(GLenum texunit, GLfloat s, GLfloat t)**
  - For immediate mode
  - Setting current texture coordinates for given texture unit
  - Coordinates are used when vertex is specified
- **void glClientActiveTexture(GLenum texture)**
  - Specifies which texture unit to make active for definition of texture coordinates using glTexCoordPointer
  - For Vertex Arrays and VBO

Example – immediate mode

```
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, tex0);
glEnable(GL_TEXTURE_2D);

glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, tex1);
glEnable(GL_TEXTURE_2D);

glBegin(GL_QUADS);
  glMultiTexCoord2f(GL_TEXTURE0, 0.0f, 0.0f);
  glMultiTexCoord2f(GL_TEXTURE1, 0.0f, 0.0f);
  glVertex3f(-1.0f, -1.0f, 0.0f);

  glMultiTexCoord2f(GL_TEXTURE0, 1.0f, 0.0f);
  glMultiTexCoord2f(GL_TEXTURE1, 5.0f, 0.0f);
  glVertex3f(1.0f, -1.0f, 0.0f);

  glMultiTexCoord2f(GL_TEXTURE0, 1.0f, 1.0f);
  glMultiTexCoord2f(GL_TEXTURE1, 5.0f, 10.0f);
  glVertex3f(1.0f, 1.0f, 0.0f);

  glMultiTexCoord2f(GL_TEXTURE0, 0.0f, 1.0f);
  glMultiTexCoord2f(GL_TEXTURE1, 0.0f, 10.0f);
  glVertex3f(-1.0f, 1.0f, 0.0f);
glEnd();
```

Example – VBO

```
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, tex0);
glEnable(GL_TEXTURE_2D);

glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, tex1);
glEnable(GL_TEXTURE_2D);

glClientActiveTexture(GL_TEXTURE0);
glEnableClientState(GL_TEXTURE_COORD_ARRAY);
glBindBuffer(GL_ARRAY_BUFFER, uiVBOTexCoords0);
glTexCoordPointer(2, GL_FLOAT, 0, NULL);

glClientActiveTexture(GL_TEXTURE1);
glEnableClientState(GL_TEXTURE_COORD_ARRAY);
glBindBuffer(GL_ARRAY_BUFFER, uiVBOTexCoords1);
glTexCoordPointer(2, GL_FLOAT, 0, NULL);

glEnableClientState(GL_VERTEX_ARRAY);
glBindBuffer(GL_ARRAY_BUFFER, uiVBOVertices);
glVertexPointer(3, GL_FLOAT, 0, NULL);

glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, uiVBOTriangles);
glDrawElements(GL_QUADS, numIndices, GL_UNSIGNED_INT, NULL);
```

Multitexturing & GLSL

- Vertex shader
  - Texture coordinates are accessed using built-in attributes gl_MultiTexCoord0, gl_MultiTexCoord1,.., gl_MultiTexCoord7
  - Or use user-defined attributes for sending multiple texture coordinates
  - Send texture coordinates to vertex shader via gl_TexCoord[0],..., gl_TexCoord[7] or user-defined varying
- Fragment shader
  - sampler2D uniform holds number of texture unit, whose actual texture will be used in texture2D function

GLSL example

```
// VERTEX SHADER
void main()
{
   gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
   gl_TexCoord[0] = gl_MultiTexCoord0;
   gl_TexCoord[1] = gl_MultiTexCoord1;
}

// FRAGMENT SHADER
uniform sampler2D tex0;
uniform sampler2D tex1;
void main()
{
   vec4 color0 = texture2D(tex0, gl_TexCoord[0].st);
   vec4 color1 = texture2D(tex1, gl_TexCoord[1].st);
   gl_FragColor = clamp(color0 + color1, 0, 1);
}

// OPENGL PROGRAM
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, tex0);
int tex0_location = glGetUniformLocation(g_programObj, "tex0");
glUniform1i(tex0_location, 0);
glEnable(GL_TEXTURE_2D);

glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, tex1);
int tex1_location = glGetUniformLocation(g_programObj, "tex1");
glUniform1i(tex1_location, 1);
glEnable(GL_TEXTURE_2D);
```

Compressed textures

- Classical textures are stored in graphics memory in raw format
- Texture data fill most significant part of graphics memory
- Compressed textures can be loaded as raw data an compressed on graphics card or loaded straight as compressed data

OpenGL formats

- Formats for compressed textures
- S3 Texture Compression
  - Extension EXT_texture_compression_s3tc
  - DXT1, DXT3, DXT5 formats – define internal format of texture
    - GL_COMPRESSED_RGB_S3TC_DXT1_EXT, 6:1 ratio
    - GL_COMPRESSED_RGBA_S3TC_DXT1_EXT, 6:1 ratio
    - GL_COMPRESSED_RGBA_S3TC_DXT3_EXT, 4:1 ratio
    - GL_COMPRESSED_RGBA_S3TC_DXT5_EXT, 4:1 ratio
- BPTC (BC7) Texture Compression
  - Extension ARB_texture_compression_bptc
  - Use COMPRESSED_RGBA_BPTC_UNORM_ARB for internal format of texture, 3:1 ratio

Compressed data

- Providing compressed texture data
- **void glCompressedTexImage2D(GLenum target, GLint level, GLenum internalformat, GLsizei width, GLsizei height, GLint border, GLsizei imageSize, const GLvoid* data)**
  - target – GL_TEXTURE_2D
  - level – mip-map level
  - internalformat – format of input and internal data
  - width, height – dimension of image
  - border – must be 0
  - imageSize – size of image data in bytes
  - data – pointer to texture data

Compressed textures

- Stored usually using .dds file format
- Encoders
  - http://code.msdn.microsoft.com/windowsdesktop/BC6HBC7-DirectCompute-35e8884a
  - http://code.google.com/p/nvidia-texture-tools/
- Loaders
  - http://www.g-truc.net/post-project-gli.html
  - http://titania-x3d.googlecode.com/git/Papers/ARB_texture_compression.pdf

Cube texture

- Texture that wraps together 6 2D textures
- Used for representation of 360 views
- Used for sky, distant background representation
- Extension ARB_texture_cube_map
- Adding new constants for glTexImage2D, glBindTexture, glEnable functions
- Using 3D texture coordinates form mapping

Cube texture example

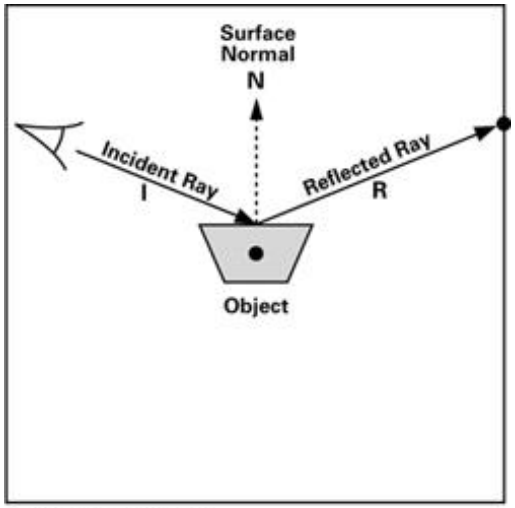
```
GLuint idSkyTexture;
glGenTextures(1, &idSkyTexture);
glBindTexture(GL_TEXTURE_CUBE_MAP, idSkyTexture);

glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_R, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);

glEnable(GL_TEXTURE_CUBE_MAP);

glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_X, 0, GL_RGBA8, CUBE_MAP_SIZE,
    CUBE_MAP_SIZE, 0, GL_RGBA, GL_UNSIGNED_BYTE, CubeMapData[0]);
glTexImage2D(GL_TEXTURE_CUBE_MAP_NEGATIVE_X, 0, GL_RGBA8, CUBE_MAP_SIZE,
    CUBE_MAP_SIZE, 0, GL_RGBA, GL_UNSIGNED_BYTE, CubeMapData[1]);
glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_Y, 0, GL_RGBA8, CUBE_MAP_SIZE,
    CUBE_MAP_SIZE, 0, GL_RGBA, GL_UNSIGNED_BYTE, CubeMapData[2]);
glTexImage2D(GL_TEXTURE_CUBE_MAP_NEGATIVE_Y, 0, GL_RGBA8, CUBE_MAP_SIZE,
    CUBE_MAP_SIZE, 0, GL_RGBA, GL_UNSIGNED_BYTE, CubeMapData[3]);
glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_Z, 0, GL_RGBA8, CUBE_MAP_SIZE,
    CUBE_MAP_SIZE, 0, GL_RGBA, GL_UNSIGNED_BYTE, CubeMapData[4]);
glTexImage2D(GL_TEXTURE_CUBE_MAP_NEGATIVE_Z, 0, GL_RGBA8, CUBE_MAP_SIZE,
    CUBE_MAP_SIZE, 0, GL_RGBA, GL_UNSIGNED_BYTE, CubeMapData[5]);
```

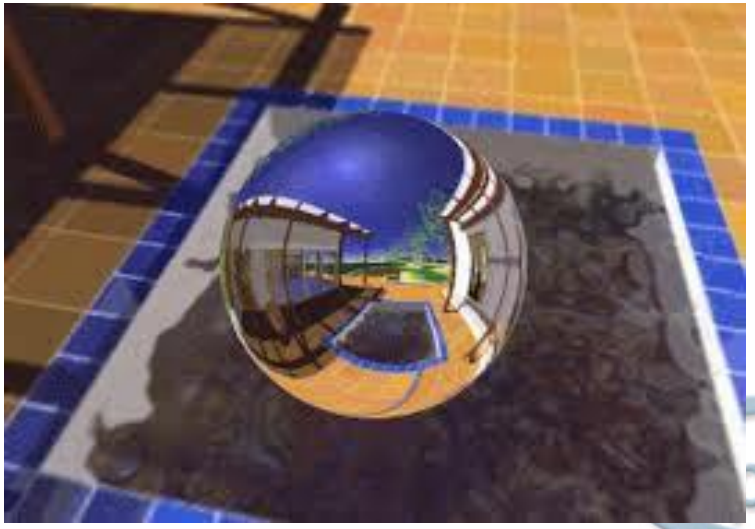
GLSL cubemap sky shaders

```
// RENDERING SKYBOX IN THE BACKGROUND AS CUBE

// VERTEX SHADER
void main()
{
   gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
   // Using coordinates of vertex also as texture coordinates
   gl_TexCoord[0] = gl_Vertex;
}

// FRAGMENT SHADER
uniform samplerCube cube_texture;

void main()
{
   // in textureCube, one of six 2D textures is picked using largest absolute texture
      coordinate, then other two coordinates are used to lookup into texture
   gl_FragColor = textureCube(cube_texture, vec3(gl_TexCoord[0]));
}
```

Environment mapping

- Treating surface of object as perfect mirror - reflecting distant static scene on surface
- Using cube texture for representing background scene
- Computing reflected view vector by normal an looking up into cube map – in world space





GLSL environment mapping

```
// VERTEX SHADER
// view matrix, model matrix is in gl_ModelViewMatrix
uniform mat4 view_matrix;
// camera position in world coordinates
uniform vec4 camera_pos;

varying vec4 V_eye;
varying vec4 L_eye;
varying vec4 N_eye;
varying vec4 RV_world;

void main(void)
{
  V_eye = view_matrix * gl_ModelViewMatrix * gl_Vertex;
  L_eye = gl_LightSource[0].position - V_eye;
  N_eye = vec4(view_matrix * gl_NormalMatrix * gl_Normal, 1.0);
  V_eye = -V_eye;

  vec4 V_world = normalize(camera_world - gl_ModelViewMatrix * gl_Vertex);
  vec4 N_world = vec4(normalize(gl_NormalMatrix * gl_Normal), 1.0);
  RV_world = reflect(-V_world, N_world);

  gl_TexCoord[0] = gl_MultiTexCoord0;
  gl_Position = gl_ProjectionMatrix * (-V_eye);
}
```

```
// FRAGMENT SHADER
varying vec4 V_eye;
varying vec4 L_eye;
varying vec4 N_eye;
varying vec4 RV_world;
uniform samplerCube texEnv;

void main(void)
{
  vec4 V = normalize(V_eye);
  vec4 L = normalize(L_eye);
  vec4 N = normalize(N_eye);

  float diffuse = clamp(dot(L, N), 0.0, 1.0);
  vec4 R = reflect(-L, N);
  float specular = pow(clamp(dot(R, V), 0.0, 1.0), 16.0);

  vec4 color = gl_LightSource[0].ambient * gl_FrontMaterial.ambient;
  color += diffuse * gl_LightSource[0].diffuse * textureCube(texEnv, vec3(RV_world));
  color += specular * gl_LightSource[0].specular * gl_FrontMaterial.specular;

  gl_FragColor = color;
}
```

The End!

Questions?