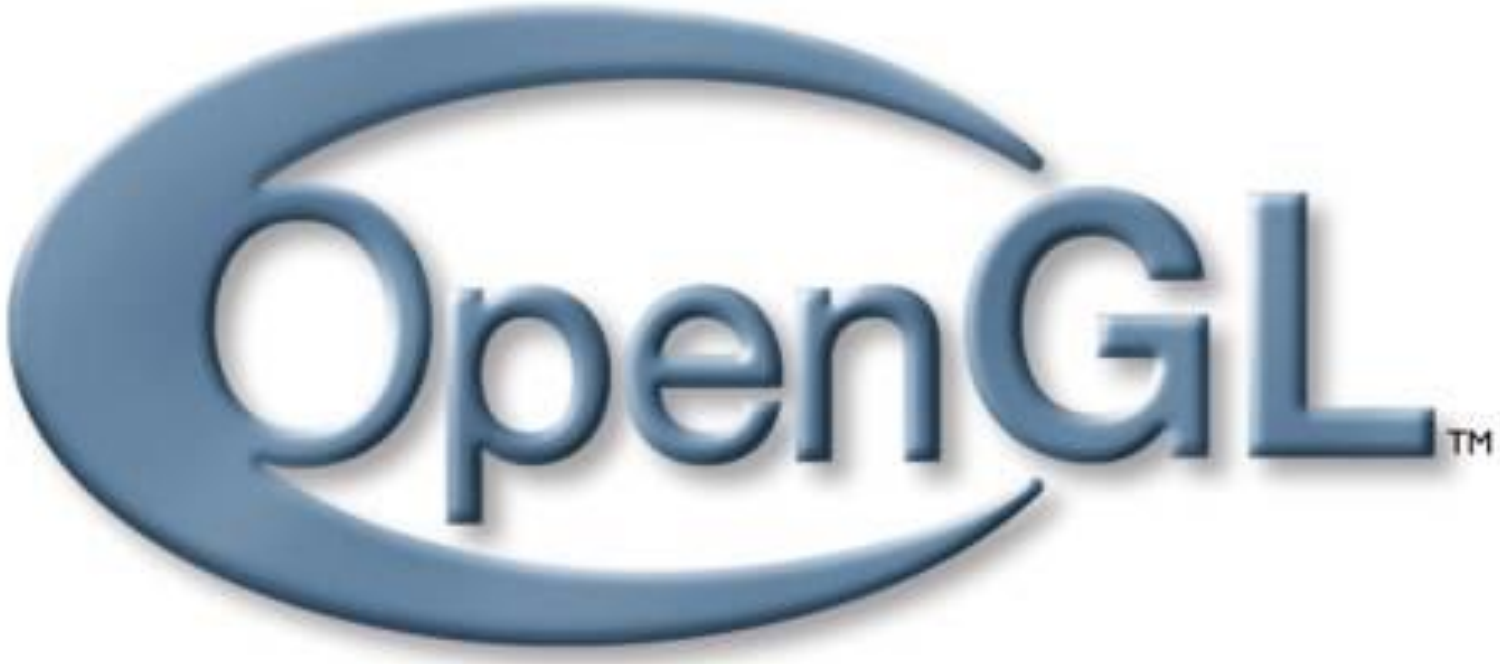
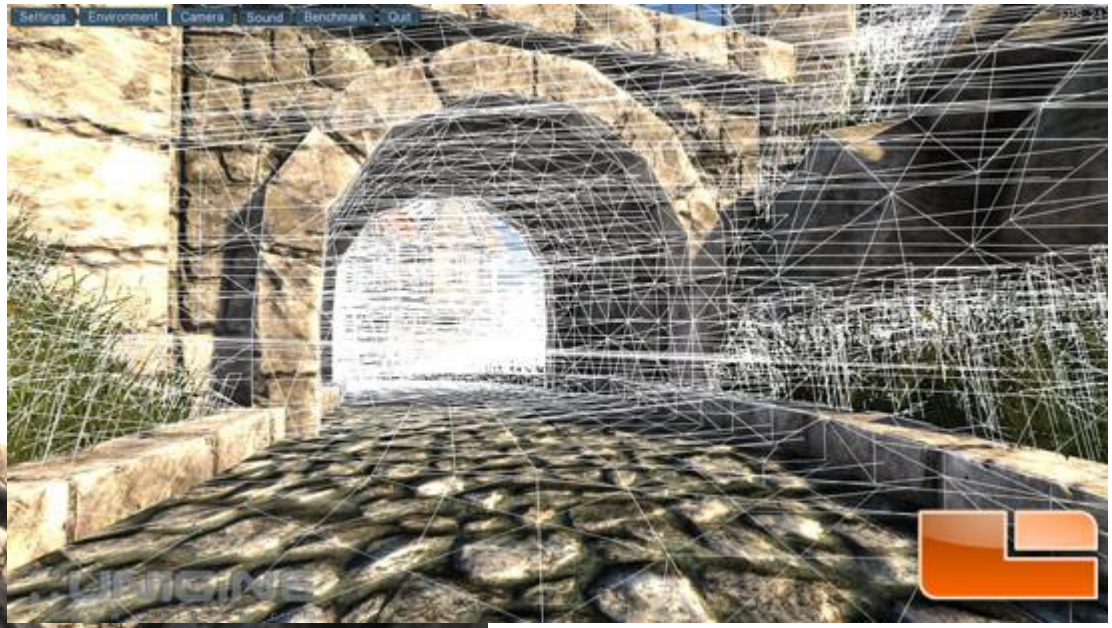


part 2

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

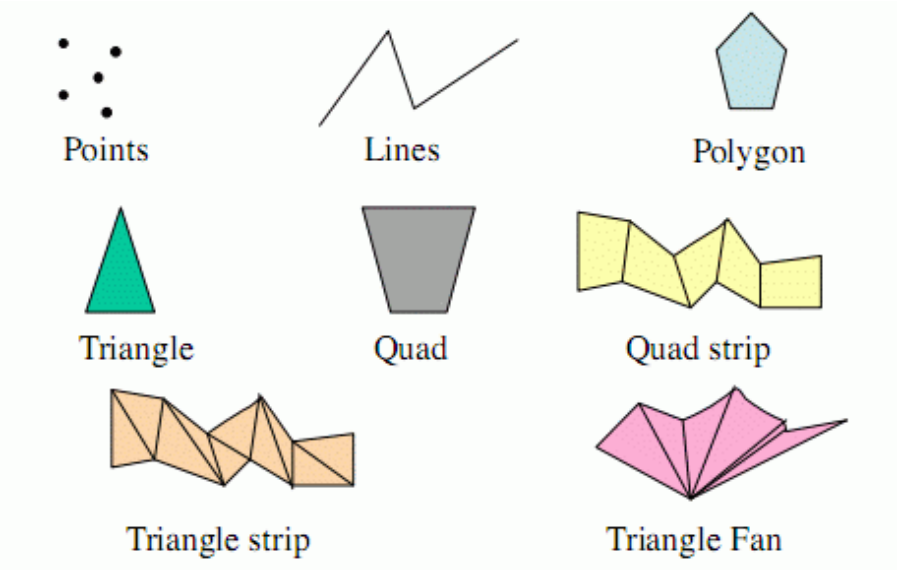
Scene geometry



OpenGL®

OpenGL basic primitives

- Simple geometry used for rendering
- Points, segments, simple convex polygons
- Each primitive given by set of vertices
  - Point – 1 vertex
  - Segment – 2 vertices
  - Triangle – 3 vertices
  - Quad – 4 vertices
  - …
- Use triangles



Vertex

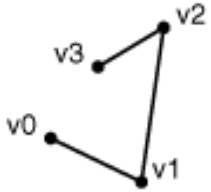
- Foundation of whole OpenGL geometry
- Containing many attributes – information, data of vertex
- Position, coordinates – 4 real numbers **(x, y, z, w)**
- Color – 4 real numbers **(r, g, b, a)**
- Normal – 3 real numbers **(x, y, z)**
- Texture coordinates – 4 real numbers **(s, t, r, q)**
- Other, user defined

OpenGL primitives

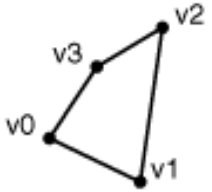
- Order of vertices defines geometric primitives (OpenGL drawing modes)

GL_POINTS

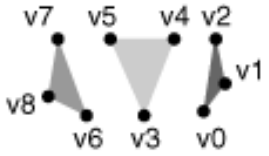
GL_LINES



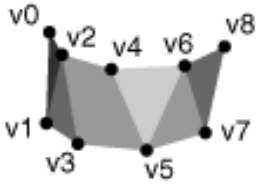
GL_LINE_STRIP



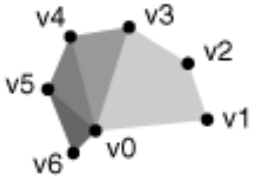
GL_LINE_LOOP



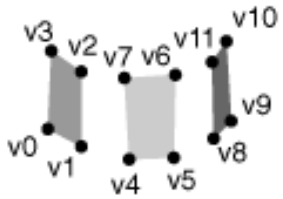
GL_TRIANGLES



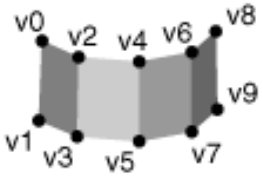
GL_TRIANGLE_STRIP



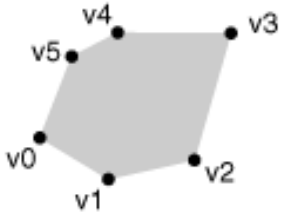
GL_TRIANGLE_FAN



GL_QUADS



GL_QUAD_STRIP



GL_POLYGON

Immediate mode

- Vertex data are given in begin-end block, each block represents one drawing mode
- For each vertex and its attribute, we have separate OpenGL function
- Inside block, define attributes of vertices
- Start of block – **glBegin(GLenum mode)**
  - mode = GL_POINTS, GL_LINES, GL_TRIANGLES, …
- End of block – **glEnd()**
  - This function does actual rendering of primitives

Immediate mode

- Specifying vertex attributes
- Color - **void glColor{34}{b s i f d ub us ui](}[v](TYPE color)**
  - Setting state variable called actual color
  - RGB or RGBA mode
  - Interval for floats - <0,1>, for bytes - <0,255>
- Normal - **void glNormal3{b s i f d](}[v](TYPE coords)**
  - Setting state variable called actual normal
- Texture coordinates - **void glTexCoord{1234}{sifd}[v](TYPE coords)**
  - Setting state variable called actual tex. coords

Immediate mode

- When specifying vertex position, all other vertex attributes are read from state variables and added to the vertex
- Position – **void glVertex{234}{sifd}[v](TYPE coords)**
  - Not setting state variable, the vertex position and other actual vertex attributes are send immediately to graphics card
- Order of vertices matters for drawing mode and for orientation of polygon

Immediate mode example

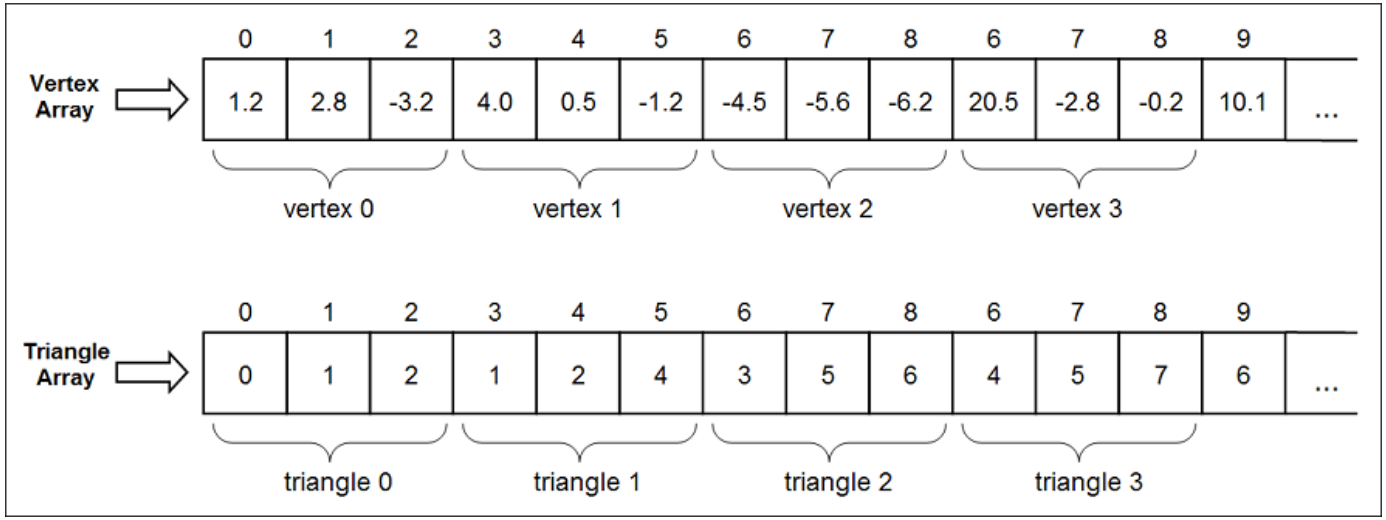
```
glBegin(GL_TRIANGLES);

// first vertex – red vertex
glColor3f(1.0f, 0.0f, 0.0f);
glNormal3f(0.0f, 1.0f, 0.0f);
glVertex3f(10.0f, 0.0f, 0.0f);

// second vertex – red vertex
glColor3f(0.0f, 1.0f, 0.0f);
glNormal3f(1.0f, 0.0f, 0.0f);
glVertex3f(0.0f, 10.0f, 0.0f);

// third vertex – red vertex
glColor3f(0.0f, 0.0f, 1.0f);
glNormal3f(0.0f, 0.0f, 1.0f);
glVertex3f(10.0f, 10.0f, 0.0f);

glEnd();
```

Vertex Arrays

- Immediate mode
  - Not very efficient, too many OpenGL calls
  - Driver must wait for glEnd and then render
  - Not used in OpenGL ES, Web GL
  - Not consistent data, lots of vertices belongs to more than one triangle
  - Cube – processing of 24 vertices
- Vertex arrays
  - From OpenGL 1.1
  - One OpenGL call for whole set of vertex attributes
  - One big array for colors, one big array for normals, …
  - One array for indices that points to vertex attr. arrays
  - Using OpenGL drawing modes
  - Cube – processing of 8 vertices

Vertex Arrays

- Proper arrays must be prepared with consistent data
- Arrays with attributes must have same number of vertices
- Specifying arrays each frame, arrays must remain allocated in processor memory



Specifying vertex attributes

- void glVertexPointer(GLint size, GLenum type, GLsizei stride, const GLvoid *pointer)
- void glColorPointer(GLint size, GLenum type, GLsizei stride, const GLvoid *pointer)
- void glNormalPointer(GLenum type, GLsizei stride, const GLvoid *pointer)
- void glTexCoordPointer(GLint size, GLenum type, GLsizei stride, const GLvoid *pointer)
- void glEdgeFlagPointer(GLsizei stride, const GLvoid *pointer)

Specifying vertex attributes

- size – number of coordinates for each record
- type – type of each coordinate in the array
- stride – number of bytes between two consecutive attributes
- pointer – pointer to data stored in processor memory
- Missing specification of the total number of bytes of whole array

Specifying vertex attributes

Command	Sizes	Type
glVertexPointer	2,3,4	GL_SHORT, GL_INT, GL_FLOAT, GL_DOUBLE
glNormalPointer	3	GL_BYTE, GL_SHORT, GL_INT, GL_FLOAT, GL_DOUBLE
glColorPointer	3,4	GL_BYTE, GL_UNSIGNED_BYTE, GL_SHORT, GL_UNSIGNED_SHORT, GL_INT, GL_UNSIGNED_INT, GL_FLOAT, GL_DOUBLE
glTexCoordPointer	1,2,3,4	GL_UNSIGNED_BYTE, GL_SHORT, GL_INT, GL_FLOAT, GL_DOUBLE
glEdgeFlagPointer	1	No type, must be GLboolean

Enabling arrays

- Enabling attribute array
  - **void glEnableClientState(GLenum par)**
  - par = GL_COLOR_ARRAY, GL_VERTEX_ARRAY, GL_NORMAL_ARRAY, GL_TEXTURE_COORD_ARRAY, GL_EDGE_FLAG_ARRAY
- Disabling attribute array
  - **void glDisableClientState(GLenum par)**
- By default, all arrays are disabled

Drawing with vertex arrays

- Specifying one vertex and its attributes
- **void glArrayElement(GLint ith)**
- Obtains data of i-th vertex from all enabled arrays
- Calls all corresponding OGL functions (glColor, glNormal, …), glVertex as last
- Calling between glBegin and glEnd
- Not recommended, too many OGL calls
- Be aware of proper index, do not point into non-allocated area!!!

Drawing with vertex arrays

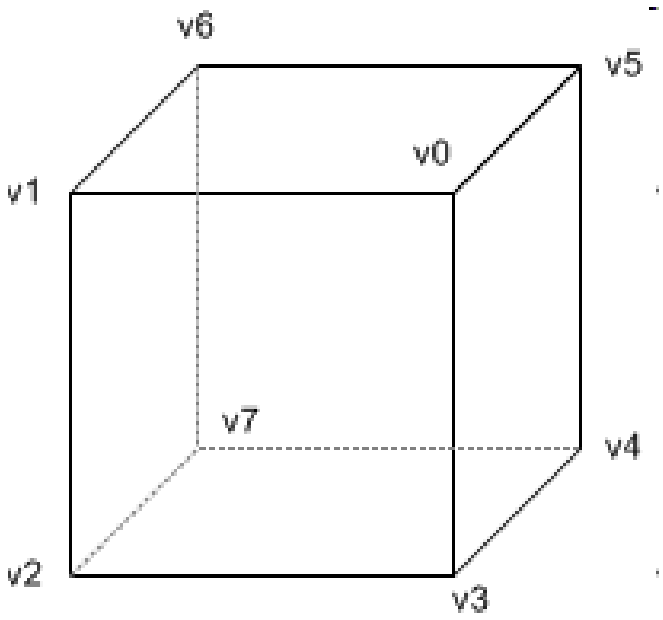
- Specyfing all indices of vertices in one array, order of indices id important
- **void glDrawElements(GLenum mode, GLsizei count, Glenum type, void *indices)**
- mode - type of primitive to construct (GL_LINES, GL_TRIANGLES, …)
- count - number of indices in array
- type - type of each indices, must be GL_UNSIGNED_BYTE, GL_UNSIGNED_SHORT, or GL_UNSIGNED_INT
- indices – pointer to array of indices

Drawing with vertex arrays

- Specyfing one special array of indices
- **void glDrawArrays(GLenum mode, GLint first, GLsizei count)**
- mode - type of drawing mode
- first - index of first element to draw
- count - number of consecutive elements
- Using array of indices
  - first,first+1,…,first+count-1
- No random access - faster

Cube example

```
GLfloat vertices[] = {1,1,1,   -1,1,1,  -1,-1,1,   1,-1,1,          // v0-v1-v2-v3
                      1,-1,-1,  1,1,-1,  -1,1,-1,  -1,-1,-1};      // v4-v5-v6-v7
GLfloat colors[] = {1,1,1,   0,1,1,  0,0,1,  1,0,1,               // c0-c1-c2-c3
                    1,0,0,   1,1,0,   0,1,0,  0,0,0};              // c4-c5-c6-c7
GLubyte indices[] = {0,1,2,3,  0,3,4,5,  0,5,6,1,                  // f0-f1-f2
                     1,6,7,2,   7,4,3,2,  4,7,6,5};                // f3-f4-f5

// activate and specify pointers to vertex arrays
glEnableClientState(GL_VERTEX_ARRAY);
glVertexPointer(3, GL_FLOAT, 0, vertices);
glEnableClientState(GL_COLOR_ARRAY);
glColorPointer(3, GL_FLOAT, 0, colors);

// draw a cube
glDrawElements(GL_QUADS, 24, GL_UNSIGNED_BYTE, indices);

// deactivate vertex arrays after drawing
glDisableClientState(GL_VERTEX_ARRAY);
glDisableClientState(GL_COLOR_ARRAY);
```



Interleaved arrays

- Combination of attributes in one array
- **void glInterleavedArrays(GLenum format, GLsizei stride, void *pointer)**
- format - specifies which attributes are used in given array and also enabled
- stride - byte offset between data start of two consecutive vertices, 0 means there is no gap between data
- pointer – pointer to a data

Interleaved arrays

- Name of format describes enabled array, number and type of coordinates
- GL_V2F, GL_V3F, GL_C4UB_V2F, GL_C4UB_V3F, GL_C3F_V3F, GL_N3F_V3F, GL_C4F_N3F_V3F, GL_T2F_V3F, GL_T4F_V4F, GL_T2F_C4UB_V3F, GL_T2F_C3F_V3F, GL_T2F_N3F_V3F, GL_T2F_C4F_N3F_V3F, GL_T4F_C4F_N3F_V4F

Vertex Buffer Objects

- Vertex arrays (VA)
  - Stored in processor memory
  - Each frame, data are copied on graphics card
- Vertex Buffer Objects (VBO)
  - Extension of VA
  - All arrays are stored in memory inside graphics card – no need for bus transfer
  - Array in graphics memory = buffer objects
  - Special management of buffer objects
  - From OpenGL 1.5

VBO management

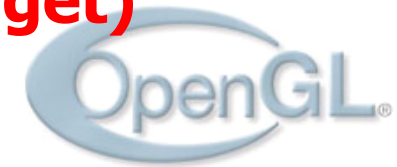
- Each buffer object has its own identifier
- Identifier is integer number
- Many buffers possible, but at any time only one is active
- All functions work with active buffer
- Generating array of free identifiers
  - **void glGenBuffers(GLsizei n, GLuint* ids)**
- Setting active buffer
  - **void glBindBuffer(GLenum target, GLuint id)**
  - target
    - GL_ARRAY_BUFFER_ARB – for vertex attribute buffers
    - GL_ELEMENT_ARRAY_BUFFER_ARB – for indices buffers

VBO load data

- Loading data from proc. memory into buffer object
- **void glBufferData(GLenum target, GLsizei size, const void* data, GLenum usage)**
- target - identifies vertex or triangle data
- size – number of bytes in data
- data – pointer to data
- usage – hint for driver, how to store data in graphics memory
  - GL_STATIC_DRAW, GL_STATIC_READ, GL_STATIC_COPY
  - GL_DYNAMIC_DRAW, GL_DYNAMIC_READ, GL_DYNAMIC_COPY
  - GL_STREAM_DRAW, GL_STREAM_READ, GL_STREAM_COPY
  - "Static" means the data in VBO will not be changed (specified once and used many times), "dynamic" means the data will be changed frequently (specified and used repeatedly), and "stream" means the data will be changed every frame (specified once and used once). "Draw" means the data will be sent to GPU in order to draw (application to GL), "read" means the data will be read by the client's application (GL to application), and "copy" means the data will be used both drawing and reading (GL to GL).

VBO management

- Replace range of data into existing buffer
  - **void glBufferSubData(GLenum target, GLint offset, GLsizei size, void* data)**
- Delete buffers
  - **void glDeleteBuffers(GLsizei n, const GLuint* ids)**
- Modify buffer data by mapping to processor memory
  - **void* glMapBuffer(GLenum target, GLenum access)**
    - access - GL_READ_ONLY, GL_WRITE_ONLY, GL_READ_WRITE
  - **GLboolean glUnmapBuffer(GLenum target)**

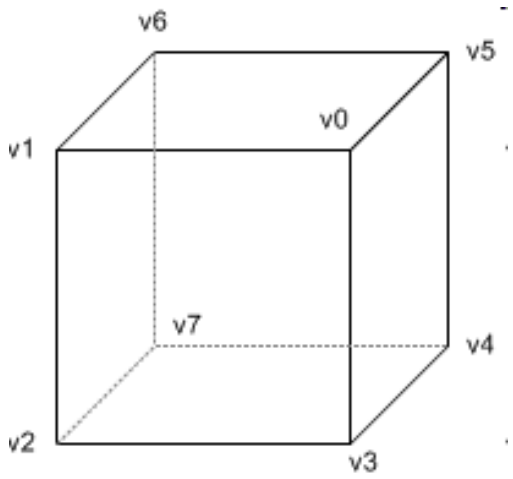


VBO drawing

- Using functionality from Vertex Arrays
- Loading data into buffers only once, then all is stored in graphics memory
- Sending NULL pointer instead of array pointer when defining arrays
- Using actual buffer as source of data for vertex attributes and indices

Cube example

```
GLfloat vertices[] = {1,1,1,   -1,1,1,  -1,-1,1,   1,-1,1,  1,-1,-1,  1,1,-1,  -1,1,-1,  -1,-1,-1};
GLfloat colors[] = {1,1,1,   0,1,1,  0,0,1,   1,0,1,  1,0,0,   1,1,0,   0,1,0,  0,0,0};
GLuint indices[] = {0,1,2,3,  0,3,4,5,  0,5,6,1,   1,6,7,2,   7,4,3,2,  4,7,6,5};

// prepare buffer for colors
unsigned int uiVBOColors;
glGenBuffers(1, &uiVBOColors);
glBindBuffer(GL_ARRAY_BUFFER, uiVBOColors);
glBufferData(GL_ARRAY_BUFFER, 8*3*sizeof(float), colors, GL_STATIC_DRAW);
// prepare buffer for coordinates
unsigned int uiVBOVertices;
glGenBuffers(1, &uiVBOVertices);
glBindBuffer(GL_ARRAY_BUFFER, uiVBOVertices);
glBufferData(GL_ARRAY_BUFFER, 8*3*sizeof(float), vertices, GL_STATIC_DRAW);
// prepare buffer with indices
unsigned int uiVBOTriangles;
glGenBuffers(1, &uiVBOTriangles);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, uiVBOTriangles);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, 6*4*sizeof(GLuint), indices, GL_STATIC_DRAW);

// activate and specify buffers to vertex attributes
glEnableClientState(GL_VERTEX_ARRAY);
glBindBuffer(GL_ARRAY_BUFFER, uiVBOVertices);
glVertexPointer(3, GL_FLOAT, 0, NULL);
glEnableClientState(GL_COLOR_ARRAY);
glBindBuffer(GL_ARRAY_BUFFER, uiVBOColors);
glColorPointer(3, GL_FLOAT, 0, NULL);

// draw a cube using buffer with indices
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, uiVBOTriangles);
glDrawElements(GL_QUADS, 24, GL_UNSIGNED_INT, NULL);

// deactivate vertex arrays after drawing
glDisableClientState(GL_VERTEX_ARRAY);
glDisableClientState(GL_COLOR_ARRAY);
```



OpenGL®

Display Lists

- Old, depreciated functionality
- Storing set of OpenGL commands as package on graphics card
- Storing matrices, raster bitmaps and images, lights, materials, textures, polygon stipples
- Easy reusability
- Possible hierarchy
- Used for font rendering
- Use VBO instead

Display List example

```
// generate id for display list
GLuint listName = glGenLists (1);

// start filling list
glNewList (listName, GL_COMPILE);
  glColor3f (1.0, 0.0, 0.0);
  glBegin (GL_TRIANGLES);
  glVertex2f (0.0, 0.0);
  glVertex2f (1.0, 0.0);
  glVertex2f (0.0, 1.0);
  glEnd ();
  glTranslatef (1.5, 0.0, 0.0);
glEndList ();

// execute all commands in list
glCallList (listName);
```

```
// use hierarchical display lists
// bicycle consists of frame,
     handlebars and 2 wheels
glNewList (bicycle,GL_COMPILE);
glCallList (handlebars);
glCallList (frame);
glTranslatef (1.0,0.0,0.0);
glCallList (wheel);
glTranslatef (3.0,0.0,0.0);
glCallList (wheel);
glEndList ();
```

Drawing rectangle

- **glRect{sifd}(T x1, T y1, T x2, T y2)**
- **glRect{sifd}v(T v1[2], T v2[2])**
- Draws full rectangle
- Affected by modelview matrix
- Accelerated version
- Same as

```
glBegin (GL_POLYGON);
  glVertex2 (x1, y1);
  glVertex2 (x2, y1);
  glVertex2 (x2, y2);
  glVertex2 (x1, y2);
glEnd ();
```

The End!

Questions?