Computer Graphics Proseminar Example Program

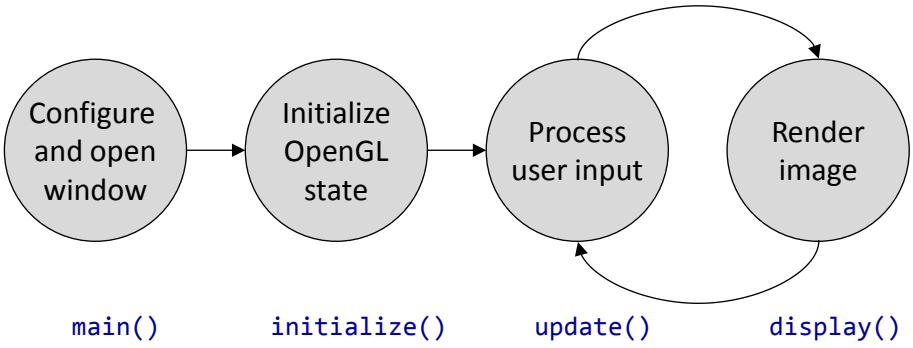
Univ.-Prof. Dr. Matthias Harders

Summer semester 2014

Content

- OpenGL program code
- GLSL syntax and code
- Analysis of example program
- First programming assignment

Standard Program Structure



Functions of OpenGL Programs

`main()`

- Program entry point
- Window and interface setup
- Call initialization routines
- Set callback functions
- Enter window main-loop

Functions of OpenGL Programs

`initialize()`

- Set OpenGL state machine
- Initialize rendering context
- Setup data for models
- Setup shader programs
- Further pre-processing

Functions of OpenGL Programs

`display()`

- Clear framebuffer
- Clear additional buffers
- Activate stored data
- Issue drawing calls

Functions of OpenGL Programs

`update()`

- Handle user input
- Update animations
- …

Steps in Shader-Based Rendering

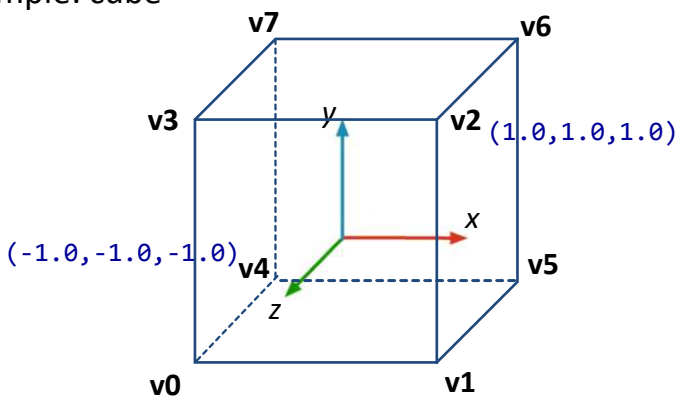
1. Create buffer objects, populate with data
2. Create shader programs
3. Associate data locations with shader variables
4. Issue render calls

Treatment of Data Objects in OpenGL

1. Create handle to buffer object (name)
   e.g. `glGenBuffers(1, &VBO);`
2. Bind buffer object to make current
   e.g. `glBindBuffer(GL_ARRAY_BUFFER, VBO);`
3. Populate buffer object with data
   e.g.
```
glBufferData(GL_ARRAY_BUFFER,
        sizeof(vertex_buffer_data),
        vertex_buffer_data, GL_STATIC_DRAW);
```
4. Unbind buffer object

Specifying Primitives for Drawing

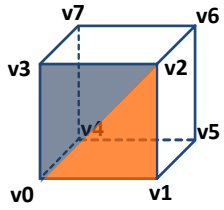
Example: cube



Option 1 – Direct Triangle List

Store triangles via three vertices each (total 36)

```
…
GLfloat vertex_buffer_data[] = {
    -1.0, -1.0,  1.0,        v0
     1.0, -1.0,  1.0,        v1
     1.0,  1.0,  1.0,        v2
     1.0,  1.0,  1.0,        v2
    -1.0,  1.0,  1.0,        v3
    -1.0, -1.0,  1.0,        v0
     1.0, -1.0,  1.0,
     1.0,  1.0,  1.0,
     1.0,  1.0, -1.0,
     1.0, -1.0,  1.0,
     1.0, -1.0, -1.0,
     1.0,  1.0, -1.0,
…
```

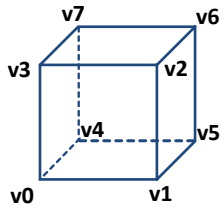


Front triangles

Option 2 – Indexed Triangle List

Store vertices & three vertex indices per triangle

```
…
GLfloat vertex_buffer_data[] = {
    -1.0, -1.0,  1.0,        v0
     1.0, -1.0,  1.0,        v1
     1.0,  1.0,  1.0,        v2
    -1.0,  1.0,  1.0,        v3
    -1.0, -1.0, -1.0,        v4
     1.0, -1.0, -1.0,        v5
     1.0,  1.0, -1.0,        v6
    -1.0,  1.0, -1.0,        v7
};
…
```

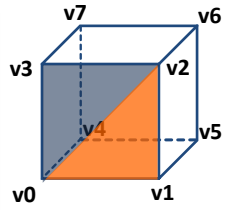


All vertices

Option 2 – Indexed Triangle List

Store all vertices, three vertex indices per triangle

```
…
GLushort index_buffer_data[] = {
        0, 1, 2,
        2, 3, 0,
        1, 5, 6,
        6, 2, 1,
        7, 6, 5,
        5, 4, 7,
        4, 0, 3,
        3, 7, 4,
        4, 5, 1,
        1, 0, 4,
        3, 2, 6,
        6, 7, 3
};   …
```



Front triangles

Setting Up Shaders in OpenGL

1. Create program
2. Create shader
3. Load shader source
4. Compile shader
5. Attach shader to program
6. Link program
7. Use program

For each shader

Issue Rendering

- Enable buffer data for rendering
  e.g. `glEnableVertexAttribArray(vPosition);`
- Specify location and format of data for rendering
  e.g. `glVertexAttribPointer(vPosition, 3, GL_FLOAT, GL_FALSE, 0, 0);`
- Issue drawing of indexed triangle list
  e.g. `glDrawElements(GL_TRIANGLES, size/sizeof(GLushort), GL_UNSIGNED_SHORT, 0);`

Alternative: draw direct triangle list

e.g. `glDrawArrays(GL_TRIANGLES, 0, numVertices);`

GLSL Data Types

- Vectors
  e.g. `vec3, vec4, ivec3, …`
- Matrices
  e.g. `mat2, mat3, mat4, …`
- Scalars
  e.g. `float, int, bool, …`
- Employ C++ style constructors:
  e.g.
  ```
  vec3 pos = vec3 (1.0, 2.0, 3.0);
  mat4 rot = mat4 (1.0);
  ```

Note: various custom initializers and constructors exist

Operators

- Standard C/C++ arithmetic and logic operators
- Operators overloaded for matrix & vector operations

e.g.
```
mat4 m;
vec4 v1,v2;
v1 = m*v2;
v2 = v1*m;
```


Components and Swizzling

- Access vector components via `[], .xyzw, .rgba, …`

e.g.
```
vec4 v1;
float x = v1.x;
float y = v1[2];
vec2 v3 = v1.xy;
mat3 m;
m[1] = vec3(0.0, 1.0, 0.0);
```

Qualifiers

- Vertex attributes/data going into shader `in`
  e.g. `in vec3 Position;`
- Vertex attributes/data going out of shader `out`
  e.g. `out vec4 vColor;`
- Read-only value from application `uniform`
  e.g. `uniform mat4 ProjectionMatrix;`
- Built-in variable `gl_Position` used for storing vertex position in clipping coordinates

Shader Layout

- Shaders require main function
- GLSL version specified in directive
- Vertex shader outputs at least to `gl_Position`
- Fragment shader outputs to custom variable

```
#version 330

uniform mat4 ProjectionMatrix;
…
void main()
{
   gl_Position = ProjectionMatrix *
       vec4(Position.x, Position.y, Position.z, 1.0);
   …
}
```

Associating Shader Variables and Data

Process after shader program linking:

- Get location of uniform variable by name

  e.g.
```
GLint projectionUniform = glGetUniformLocation
(ShaderProgram, "ProjectionMatrix");
```

- Set current value of uniform variable

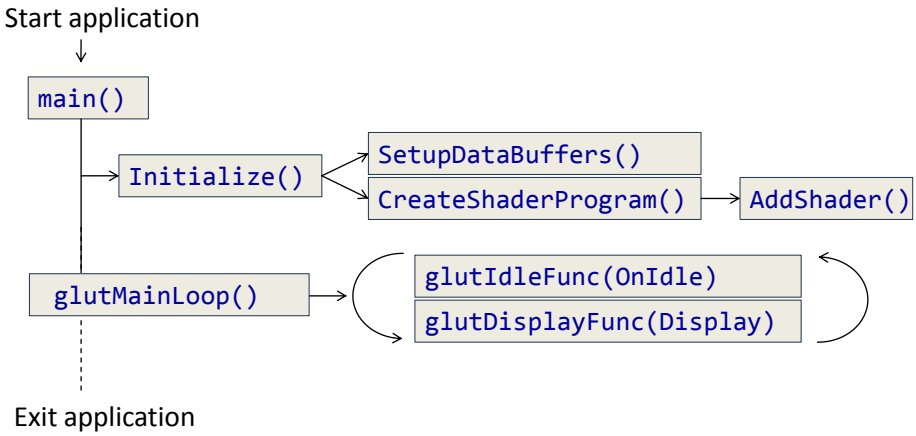
  e.g.
```
glUniformMatrix4fv (projectionUniform, 1,
GL_TRUE, ProjectionMatrix);
```


Using Shaders In OpenGL

- Compile and link shader from executable program
- OpenGL provides run-time compiler/linker
- Vertex and fragment shader mandatory
- Initiate shader execution by issuing drawing calls

  e.g.
```
glDrawArrays(GL_TRIANGLES, 0, numVertices);
```

Example Program Skeleton



Example Program – Includes

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#include <GL/glew.h>
#include <GL/freeglut.h>

#include "LoadShader.h
#include "Matrix.h"
```

OpenGL headers for GLUT and GLEW

Example Program – Includes

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#include <GL/glew.h>
#include <GL/freeglut.h>

#include "LoadShader.h
#include "Matrix.h"
```

Local headers

Example Program – Main Function

```
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH);
    glutInitWindowSize(600, 600);
    glutInitWindowPosition(400, 400);
    glutCreateWindow("CG Proseminar - Rotating Cube");
    …

    GLenum res = glewInit();
}
```

Use double buffer, RGBA, and depth buffer

Example Program – Main Function

```
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH);
    glutInitWindowSize(600, 600);
    glutInitWindowPosition(400, 400);
    glutCreateWindow("CG Proseminar - Rotating Cube");
    …

    GLenum res = glewInit();
}
```

Initialize GL Extension Wrangler

Example Program – Main Function

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

    glutIdleFunc(OnIdle);
    glutDisplayFunc(Display);
    glutMainLoop();

    …
}
```

Idle function, continuously executed

Example Program – Initialization

```
void Initialize(void)
{
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glEnable(GL_DEPTH_TEST);
    glDepthFunc(GL_LESS);
    SetupDataBuffers();
    CreateShaderProgram();
    SetIdentityMatrix(ProjectionMatrix);
    SetIdentityMatrix(ViewMatrix);
    SetIdentityMatrix(ModelMatrix);
    …
}
```

Enable depth tests (state) for visibility handling

Example Program – Setup Data

```
void SetupDataBuffers()
{
    glGenBuffers(1, &VBO);
    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertex_buffer_data),
        vertex_buffer_data, GL_STATIC_DRAW);

    glGenBuffers(1, &IBO);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, IBO);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER,
        sizeof(index_buffer_data), index_buffer_data,
        GL_STATIC_DRAW);
    …
}
```

Generate one buffer object name

Example Program – Setup Data

```
void SetupDataBuffers()
{
    glGenBuffers(1, &VBO);
    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertex_buffer_data),
        vertex_buffer_data, GL_STATIC_DRAW);

    glGenBuffers(1, &IBO);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, IBO);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER,
        sizeof(index_buffer_data), index_buffer_data,
        GL_STATIC_DRAW);
    …
}
```

Activate the buffer object

Example Program – Setup Data

```
void SetupDataBuffers()
{
    glGenBuffers(1, &VBO);
    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertex_buffer_data),
        vertex_buffer_data, GL_STATIC_DRAW);

    glGenBuffers(1, &IBO);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, IBO);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER,
        sizeof(index_buffer_data), index_buffer_data,
        GL_STATIC_DRAW);
    …
}
```

Load (vertex) data into buffer object

Example Program – Setup Data

```
void SetupDataBuffers()
{
    glGenBuffers(1, &VBO);
    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertex_buffer_data),
        vertex_buffer_data, GL_STATIC_DRAW);

    glGenBuffers(1, &IBO);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, IBO);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER,
        sizeof(index_buffer_data), index_buffer_data,
        GL_STATIC_DRAW);
    …
}
```

Data constant, used for drawing

Example Program – Setup Data

```
void SetupDataBuffers()
{
    glGenBuffers(1, &VBO);
    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertex_buffer_data),
        vertex_buffer_data, GL_STATIC_DRAW);

    glGenBuffers(1, &IBO);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, IBO);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER,
        sizeof(index_buffer_data), index_buffer_data,
        GL_STATIC_DRAW);
    …
}
```

Stored data are element indices

Example Program – Display

```
void Display()
{
    glEnableVertexAttribArray(vPosition);
    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    glVertexAttribPointer(vPosition,3,GL_FLOAT,GL_FALSE,0,0);
    …

    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, IBO);
    GLint size;
    glGetBufferParameteriv(GL_ELEMENT_ARRAY_BUFFER,
    GL_BUFFER_SIZE, &size);
    …
}
```

Enable variables in the shader

Example Program – Display

```
void Display()
{
    glEnableVertexAttribArray(vPosition);
    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    glVertexAttribPointer(vPosition,3,GL_FLOAT,GL_FALSE,0,0);
    …

    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, IBO);
    GLint size;
    glGetBufferParameteriv(GL_ELEMENT_ARRAY_BUFFER,
    GL_BUFFER_SIZE, &size);
    …
}
```

Bind buffer object to target

Example Program – Display

```
void Display()
{
    glEnableVertexAttribArray(vPosition);
    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    glVertexAttribPointer(vPosition,3,GL_FLOAT,GL_FALSE,0,0);
    …

    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, IBO);
    GLint size;
    glGetBufferParameteriv(GL_ELEMENT_ARRAY_BUFFER,
    GL_BUFFER_SIZE, &size);
    …
}
```

Tell shader variables where to find their data

Example Program – Display

```
void Display()
{
    glEnableVertexAttribArray(vPosition);
    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    glVertexAttribPointer(vPosition,3,GL_FLOAT,GL_FALSE,0,0);
    …

    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, IBO);
    GLint size;
    glGetBufferParameteriv(GL_ELEMENT_ARRAY_BUFFER,
    GL_BUFFER_SIZE, &size);
    …
}
```

Specify number of components per vertex attribute

Example Program – Display

```
void Display()
{
    glEnableVertexAttribArray(vPosition);
    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    glVertexAttribPointer(vPosition,3,GL_FLOAT,GL_FALSE,0,0);
    …

    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, IBO);
    GLint size;
    glGetBufferParameteriv(GL_ELEMENT_ARRAY_BUFFER,
    GL_BUFFER_SIZE, &size);
    …
}
```

Flags if data needs to be normalized

Example Program – Display

```
void Display()
{
    glEnableVertexAttribArray(vPosition);
    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    glVertexAttribPointer(vPosition,3,GL_FLOAT,GL_FALSE,0,0);
    …

    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, IBO);
    GLint size;
    glGetBufferParameteriv(GL_ELEMENT_ARRAY_BUFFER,
    GL_BUFFER_SIZE, &size);
    …
}
```

Return selected parameter from object

Example Program – Display

```
void Display()
{
    …
    GLint RotationUniform = glGetUniformLocation
        (ShaderProgram, "ModelMatrix");

    glUniformMatrix4fv(RotationUniform,1,GL_TRUE,ModelMatrix);

    glDrawElements(GL_TRIANGLES, size/sizeof(GLushort),
    GL_UNSIGNED_SHORT, 0);
    …
}
```

Return location of uniform shader variable

Example Program – Display

```
void Display()
{
    …
    GLint RotationUniform = glGetUniformLocation
        (ShaderProgram, "ModelMatrix");

    glUniformMatrix4fv(RotationUniform,1,GL_TRUE,ModelMatrix);

    glDrawElements(GL_TRIANGLES, size/sizeof(GLushort),
    GL_UNSIGNED_SHORT, 0);
    …
}
```

Specify value of a uniform variable

Example Program – Display

```
void Display()
{
    …
    GLint RotationUniform = glGetUniformLocation
        (ShaderProgram, "ModelMatrix");

    glUniformMatrix4fv(RotationUniform,1,GL_TRUE,ModelMatrix);

    glDrawElements(GL_TRIANGLES, size/sizeof(GLushort),
    GL_UNSIGNED_SHORT, 0);
    …
}
```

Number of matrices

Example Program – Display

```
void Display()
{
    …
    GLint RotationUniform = glGetUniformLocation
        (ShaderProgram, "ModelMatrix");

    glUniformMatrix4fv(RotationUniform,1,GL_TRUE,ModelMatrix);

    glDrawElements(GL_TRIANGLES, size/sizeof(GLushort),
    GL_UNSIGNED_SHORT, 0);
    …
}
```

Flag if matrix requires transpose

Example Program – Display

```
void Display()
{
    …
    GLint RotationUniform = glGetUniformLocation
        (ShaderProgram, "ModelMatrix");

    glUniformMatrix4fv(RotationUniform,1,GL_TRUE,ModelMatrix);

    glDrawElements(GL_TRIANGLES, size/sizeof(GLushort),
    GL_UNSIGNED_SHORT, 0);
    …
}
```

Render primitives from array data

Example Program – Display

```
void Display()
{
    …
    GLint RotationUniform = glGetUniformLocation
        (ShaderProgram, "ModelMatrix");

    glUniformMatrix4fv(RotationUniform,1,GL_TRUE,ModelMatrix);

    glDrawElements(GL_TRIANGLES, size/sizeof(GLushort),
    GL_UNSIGNED_SHORT, 0);
    …
}
```

Specifies number of elements to be rendered

Programming Assignment 1

Create a shader-based OpenGL program showing an **animated mobile** (using OpenGL3.3 and GLSL 3.3)

Programming Assignment 1

Main Tasks:

1. Setup hierarchical geometrical model; create basic 3D objects; use at least three levels
2. Add random, but plausible animations to objects
3. Use perspective projection; implement an automatically rotating as well as a user-controlled camera

Programming Assignment 1

- Develop using OpenGL 3.3, GLSL 3.3
- Programming in teams of two students allowed
- Submit code as ZIP via email (see assignment sheet)
- Code must run and compile on Linux (reference RR15)
- Deadline: April 8th, 2014, 10am

Tasks for Next Week

- Start working on first programming assignment
- Use code discussed so far as starting point
- Hints on implementation in next Proseminar