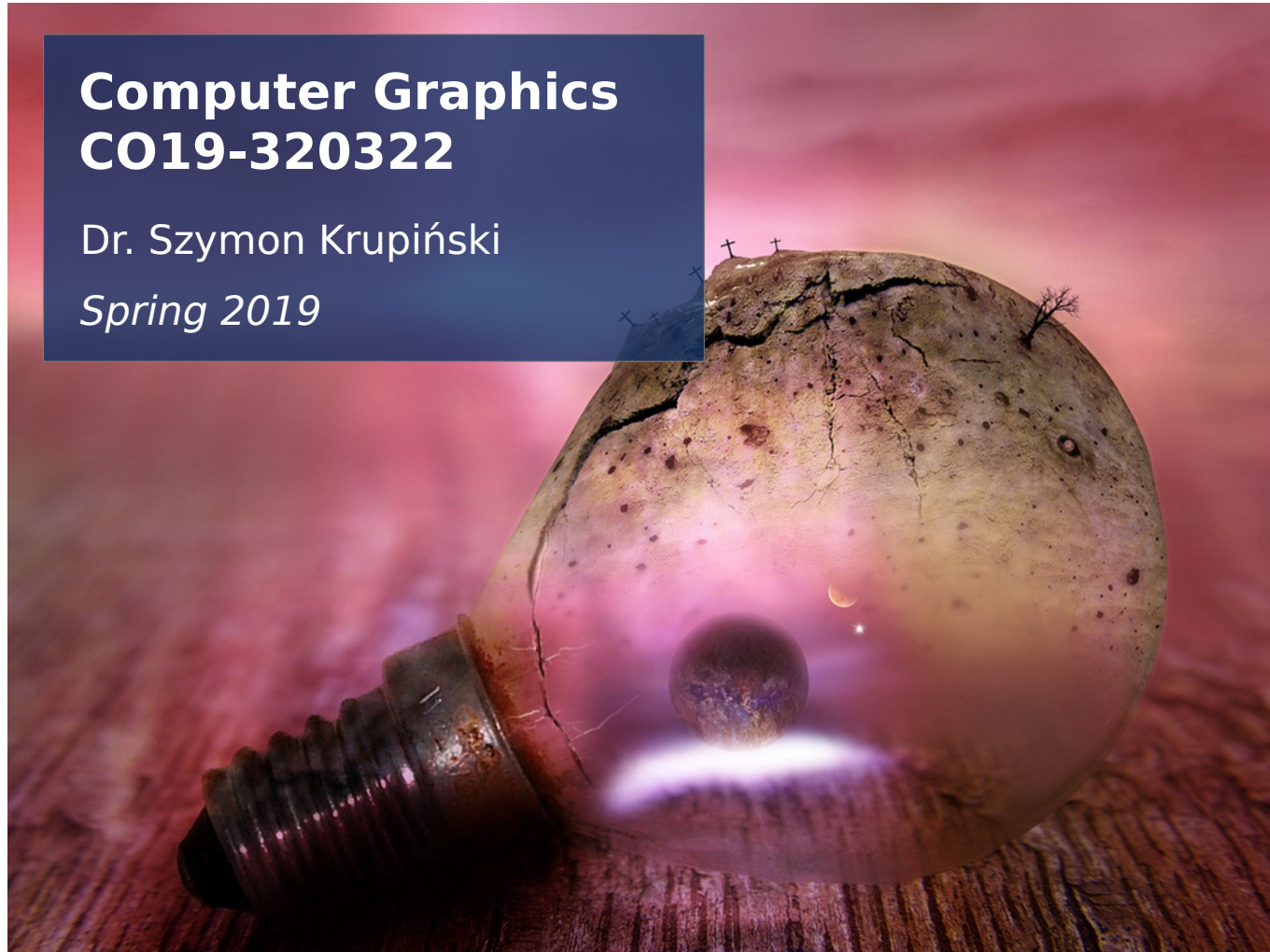


Lecture 6:

OpenGL Basics 2

Computer Graphics
CO19-320322

Dr. Szymon Krupiński
Spring 2019



Good info source

- The official documentation portal:

<https://www.khronos.org/registry/OpenGL-Refpages/gl4/>

- Older versions:

<https://www.khronos.org/registry/OpenGL-Refpages/gl2.1/>

- watch out for the interface version!
- The OpenGL 3.2 core specification removes the majority of the fixed function pipeline previously used, and replaces it with a completely programmable architecture using shaders
- “Modern CG pipeline” = (more) programmable CG pipeline



Current API Versions

- [OpenGL ES 3.2 and OpenGL ES Shading Language 3.20](#)
- [OpenGL 4.5 and OpenGL Shading Language 4.50](#)

Older API Versions

Note that each reference page in the Current Versions pages linked above includes ver versions, so (for example) the OpenGL 3.x reference pages are no longer provided. The linked here are increasingly out of date, and may eventually be removed.

The OpenGL 2.1 pages are the only source of reference material for GLX, GLU, and the be retained for that reason, even though they are otherwise useless, or we may eventu OpenGL 4.5 pages instead.

- [OpenGL ES 3.1 and OpenGL ES Shading Language 3.10](#)
- [OpenGL ES 3.0 and OpenGL ES Shading Language 3.00](#)
- [OpenGL ES 2.0](#)
- [OpenGL ES 1.1](#)
- [OpenGL 2.1](#) (including GLX, GLU, and fixed-function GL compatibility profile APIs)

Code breakdown

- Includes:

```
#include <GL/glut.h>
```

- (or, if using other members of the family:)

```
#include <GL/glu.h>
```

```
#include <GL/gl.h>
```

- For linking to the appropriate libraries:
 - gcc: specify `-lglut -lGLU -lGL`
 - MSVS: add Glut libraries in the project options

Code breakdown

- Minimalistic `main()`

```
void main(int argc, char** argv) {  
  
    glutInit(&argc, argv);  
    glutInitDisplayMode(GLUT_RGB|GLUT_DOUBLE);  
  
    glutCreateWindow(argv[0]);  
  
    // Setting up callbacks  
    glutDisplayFunc(display);  
    glutReshapeFunc(reshape);  
    glutKeyboardFunc(keyboard);  
    glutIdleFunc(idle);  
  
    // Beginning of the event processing loop  
    glutMainLoop();  
}
```

Initialisation

- Configure and open new display window

```
glutInitWindowSize(400,400);  
glutInitWindowPosition(100,100);  
glutCreateWindow("Window name");
```

The later one returns a unique handle – keep it to work with many windows

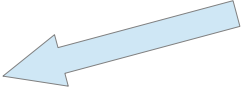
- Initialise GLUT state

```
glutInit(&argc, argv);  
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
```

- GLUT_DOUBLE | GLUT_RGB – binary *OR*ing constants to initialise a flag register
- Our choices:
 - GLUT_RGB vs GLUT_INDEX – color modes (will cover later)
 - GLUT_SINGLE vs GLUT_DOUBLE – single drawing buffer or two alternating buffers to draw to which we flip after every drawing cycle? Useful for real time animation
 - glFlush() vs glutSwapBuffers();
- Buffer enabling flags: GLUT_DEPTH, GLUT_STENCIL, GLUT_ACCUM – enable features such as the depth buffer

Callbacks

- Asynchronous user actions are managed through callbacks – user defined functions which are evoked once an event takes place.
- The function handle is given as argument (here user defined arbitrarily named `display()` and `reshape()` functions):
- `glutDisplayFunc(display);`
- `glutReshapeFunc(reshape);`
 - Prototype of the reshape callback:
`void (*func)(int width, int height)`
- Result: `display()` will be called when the window is redrawn and `reshape()` will be called if the window geometry changes



The function passed as the display callback usually does the most important: **the drawing!**

More callbacks

- `GlutKeyboardFunc()`
 - Called when a key is pressed or mouse clicked (coords relative to window)
 - `void (*func)(unsigned int key, int x, int y)`
 - Key is ascii of the keyboard button
- `GlutSpecialFunc()`
 - Like above but for special buttons (F12, ESC, etc)
- `GlutMouseFunc`
 - `void (*func)(int button, int state, int x, int y)`
 - Button: left, right, middle
 - State: pressed, released
 - x, y as above
- `GlutMotionFunc()`
 - `void (*func)(int x, int y)`
 - Gets mouse movement while one of the buttons is pressed (drag & drop)
- `GlutTimerFunc(unsigned int msecs, void (*func)(int value),value);`
 - Typical timer callback which will be triggered when the period of
- `GlutPostRedisplay()`
 - The next iteration through `glutMainLoop`, the window's display callback will be called to redisplay the window's normal plane. Multiple calls to `glutPostRedisplay` before the next display callback opportunity generates only a single redisplay callback.
- `GlutIdleFunc()`
 - What shall I do when I'm doing nothing? A way to manage the frames per second performance (FPS)

Callbacks – window resize

```
// The window resizing callback function
void changeSize(int w, int h)
{
    // Prevent a divide by zero, when window is too short
    // (you cant make a window of zero width).
    if(h == 0)
        h = 1;

    float ratio = 1.0* w / h;

    // Reset the coordinate system before modifying
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    // Set the viewport to be the entire window
    glViewport(0, 0, w, h);

    // Set the correct perspective.
    gluPerspective(90,ratio,1,1000);

    // Or maybe orthogonal projection
    // glOrtho(-5,5,-5,5,1, 100);
}
```

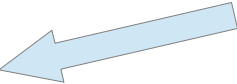

Callbacks - keyboard

```
// The keyboard callback function
void processKeys(unsigned char key, int x, int y)
{
    switch(key) {
        case 'w': theta+=0.1f; break;
        case 's': theta-=0.1f; break;
        case 'a': phi+=0.1f; break;
        case 'd': phi-=0.1f; break;
        case 't': ro+=0.3f; break;
        case 'g': ro-=0.3f; break;
        case 27: exit(0);
    }

    // Keep the values in bounds
    // NOTE: glRotate expects the angle in degrees, but here radians are used
    //       as they're used to compute sin() and cos() in order to transform
    //       the spherical coordinates into cartesian

    while (theta>2*M_PI) theta-=2*M_PI;
    while (theta<0) theta+=2*M_PI;
    while (phi>2*M_PI) phi-=2*M_PI;
    while (phi<0) phi += 2*M_PI;

    // we need to redisplay using the new position of the camera
    glutPostRedisplay();
}
```



This defines interaction with the user through the pressed keyboard buttons – we could even build a game like this!

Event processing loop

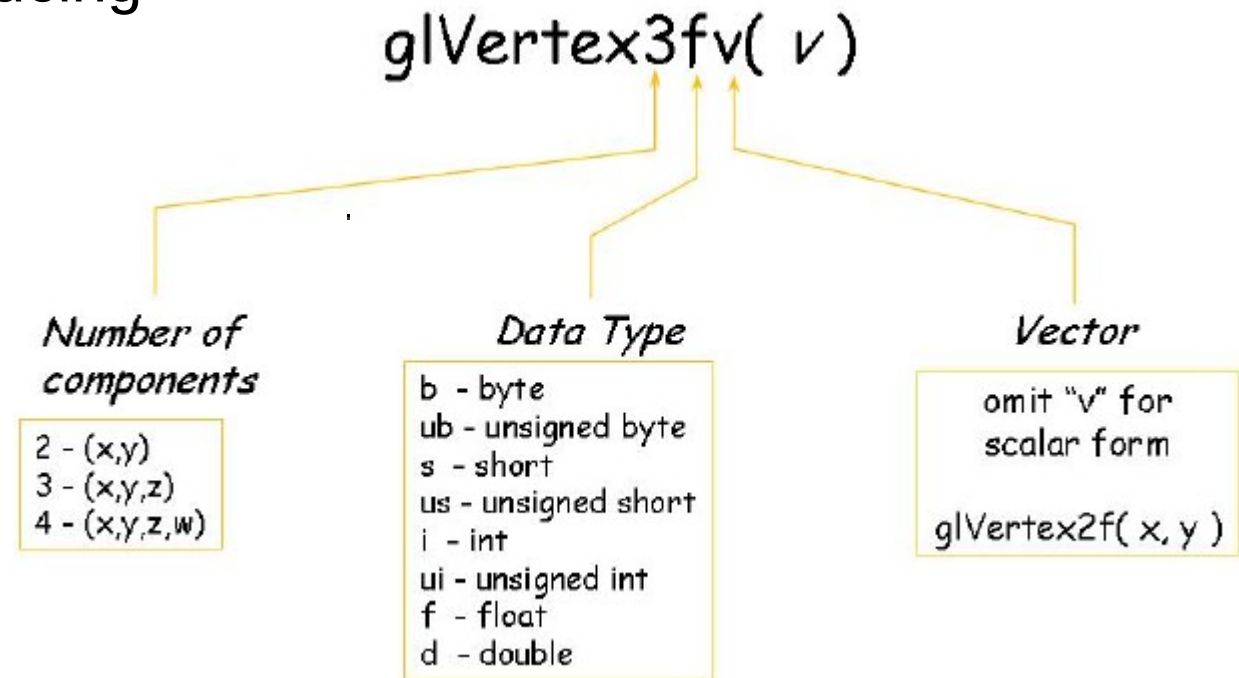
- Concept used in many libraries
- Hand over control to the library and the callbacks mechanism
- Asynchronous program starts when `glutMainLoop()` is called

Controlling state

- OpenGL has a simple interface for setting current state elements such as
 - Line type: `glLineStaple(repeat, pattern)`
 - Shade model: `glShadeModel(GL_SMOOTH)`
 - Point size: `glPointSize(size)`
- The current type of setting is used until the state is modified using another call with another parameter or another function which modifies the state is called

Naming conventions

- OpenGL has a simple set of naming rules
 - Prefixes: which sub-library the function comes from
 - In case of function accepting/producing vectors or arrays:
 - number of components
 - number type
 - if v is present: vector



Scene setup

- Before any meaningful rendering, the scene has to be configured
 - Object
 - Camera
 - Lighting
 - (Materials, shaders, ...)
- The elements above must be in a good spatial relation
 - Camera frustrum enclosing what we want to render
 - Lighting pointing towards and of sufficient intensity to brighten the object
- Some of the optional mechanisms must be explicitly enabled

```
glEnable(GL_LIGHT0);  
glEnable(GL_LIGHT1);  
glEnable(GL_LIGHTING);
```

Projection and camera setup

- `glViewport(GLint x, GLint y, GLsizei width, GLsizei height)`
 - Defines a pixel rectangle in the window into which the final image is mapped. By default: the whole open window
- `glDepthRange(GLclampd near, GLclampd far)`
 - way to delimit minimum and maximum depth coordinate after projection

Projection setup

- Example using the old style of processing (fixed function pipeline)

```
glMatrixMode(GL_PROJECTION)
```

```
gluPerspective(GLdouble fovy, GLdouble aspect, GLdouble  
zNear, GLdouble zFar);
```


Camera setup

- Moving the camera around analogy: “place the tripod, aim the camera”

```
gluLookAt( eyex, eyey, eyez,  
           aimx, aimy, aimz,  
           upx, upy, upz)
```

(“up” vector determines unique orientation)

Managing matrix stacks

- Warning! It's an old mechanism, it is deprecated/removed in newer versions of OpenGL
- We have shown in the course that so many transformations can be represented as a number of consecutive matrix multiplications! In OpenGL they are called matrix stacks
- `glMatrixMode(GL_PROJECTION);`
 - Define which stack we will affect with subsequent calls
 - `GL_MODELVIEW` position and orientation of the 3D objects in the scene
 - `GL_PROJECTION` manipulating the projection
 - `GL_TEXTURE` useful for transforming textures (we haven't yet covered this one!)
 - `GL_COLOR` (with `ARB_imaging` extension)
- `glLoadIdentity();`
 - Initialises to identity

Managing matrix stacks

- The matrices manipulated by all operations discussed so far are the ones that sit at the top of the matrix stack

`glPushMatrix()`, `glPopMatrix()`

- When a matrix is pushed on the top of the matrix stack, it simply duplicates the current top matrix
- When popping a matrix, the topmost one is deleted

Managing matrix stacks

All these transformations work on the topmost matrix:

```
GLfloat xRotated, yRotated, zRotated;
// angles by which the scene will be rotated
xRotated = yRotated = zRotated = 30.0;
// make the top the identity matrix.
glLoadIdentity();
// traslate the drawing plane by z = -4.0
glTranslatef(0.0,0.0,-10.0);
// changing in transformation matrix.
// rotation about X axis
glRotatef(xRotated,1.0,0.0,0.0);
// rotation about Y axis
glRotatef(yRotated,0.0,1.0,0.0);
// rotation about Z axis
glRotatef(zRotated,0.0,0.0,1.0);
// scaling transformation
glScalef(1.0,1.0,1.0);
```

Modern way?

- The alternative to this is at this moment not yet accessible to us as it is based heavily on shaders!

Your TAs

They will help you with your questions and installation

- Majorka

m.thanasi@jacobs-university.de

- Muhammad

muh.hassan@jacobs-university.de

Thank you!

- Questions?

