About me

- PhD student in Image Analysis (since 2011)
- My research focuses on developing methods and tools for interactive analysis of medical 3D images
Graphics programming

- This lecture will cover the basics of graphics programming with C++ and OpenGL.
- It will only scratch the surface of the subject, but you can find more extensive details about OpenGL in the course book or at http://www.opengl.org/documentation/
- Getting proficient with OpenGL requires a lot of practice, so try to solve as many exercises from the book as possible.
OpenGL

- A cross-platform, low-level API for rendering 2D and 3D graphics.
- The first version of the API was released in 1992. The specification is currently managed by the Khronos Group.
- Easy to use and provides excellent rendering performance. Suitable for real-time rendering.
- Defined as a finite state machine.
- Omits windowing and input to avoid window system dependencies.
Output primitives

- The basic building blocks of graphics. Examples:
  - points
  - lines
  - polygons
  - circles
  - character strings
Attributes

• Defines properties of output primitives. Examples:
  - intensity
  - color
  - line style
  - text style
  - area-filling pattern
The OpenGL graphics pipeline (recap)

- The input is the vertices (and their attributes) of the object to be rendered
- The output are pixels on the screen
- Performs
  - Transformations
  - Clipping
  - Shading and illumination
  - Texturing, etc
Coordinate systems

- The OpenGL graphics pipeline defines six different coordinate systems:
  - Object (or model) coordinates
  - World coordinates
  - Eye (or camera) coordinates
  - Clip coordinates
  - Normalized device coordinates
  - Window (or screen) coordinates
- Coordinate systems and transformations are covered in more details in lectures 3, 4, and 8
Example program – triangle.cpp

- A simple OpenGL program that draws a white triangle on a black background.
Source code

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

void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_TRIANGLES);
        glVertex3f(-0.5, -0.5, 0.0);
        glVertex3f(0.5, -0.5, 0.0);
        glVertex3f(0.0, 0.5, 0.0);
    glEnd();
    glFlush();
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutCreateWindow("Triangle");
    glutDisplayFunc(display);
    glutMainLoop();
}
```
Compilation

- On Linux, this program can be compiled with the command
  
  `gcc triangle.cpp -lglut -lGL`
GLUT

- The OpenGL Utility Toolkit (GLUT) is a simple and highly portable toolkit that provides an interface between the graphics system and the underlying operating and window system.
- It allows the user to create and manage windows, monitor input devices, create pop-up menus, etc.
- In triangle.cpp, we use GLUT to set up a window and an OpenGL context.
Event loop

- Note that the program defines a display callback function named `display`
- Every GLUT program must have a display callback
- The display callback is executed whenever OpenGL decides the display must be refreshed, e.g., when the window is opened or resized
- The main function ends with the program entering an event loop
Defaults

- triangle.cpp is very simple
- It makes heavy use of state variable default values for
  - Viewing
  - Colors
  - Window parameters
- You will learn how to configure the parameters yourself during the labs (and when you work through the exercises in the book)
OpenGL primitives

- The following geometric primitives are the basic building blocks used in OpenGL applications:
  - Points: `GL_POINTS`
  - Lines: `GL_LINES`, `GL_LINE_STRIP`, `GL_LINE_LOOP`
  - Triangles: `GL_TRIANGLES`, `GL_TRIANGLE_STRIP`, `GL_TRIANGLE_FAN`
  - Quadrilaterals and other polygons: `GL_QUADS`, `GL_QUAD_STRIP`, `GL_POLYGON`
Vertex attributes

• OpenGL provides functions for specifying certain attributes of vertices. Examples:
  - `glColor` – sets the current vertex color
  - `glNormal` – sets the current normal vector
  - `glTexCoord/glMultiTexCoord` – sets the current texture coordinates
• We will look more on these later.
Interpolation of vertex attributes

- Vertex attributes are (unless otherwise specified) linearly interpolated across geometric primitives. Hence, if we define a triangle such that each of its vertices has a different color, the result will look like this:
Source code

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

void display(){
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_TRIANGLES);
    glColor3f(1.0, 0.0, 0.0);
    glVertex3f(-0.5, -0.5, 0.0);
    glColor3f(0.0, 1.0, 0.0);
    glVertex3f(0.5, -0.5, 0.0);
    glColor3f(0.0, 0.0, 1.0);
    glVertex3f(0.0, 0.5, 0.0);
    glEnd();
    glFlush();
}

int main(int argc, char** argv){
    glutInit(&argc, argv);
    glutCreateWindow("Triangle");
    glutDisplayFunc(display);
    glutMainLoop();
}
```
In OpenGL, there are usually multiple functions for a given logical function. Example:

- `glVertex3f(GLfloat x, GLfloat y, GLfloat z)`
- `glVertex3i(GLint x, GLint y, GLint z)`
- `glVertex4f(GLfloat x, GLfloat y, GLfloat z, GLfloat w)`
- `glVertex2f(GLfloat x, GLfloat y)`

For vertices or vertex attributes, the underlying storage mode is the same. For instance, vertices are represented as 4-element vectors, so a call to `glVertex2f` will actually tell OpenGL to create a 4-element vector (with the z and w components set to default values).
RGB color in OpenGL

- Each color component is stored separately in the frame buffer
- Usually 8 bits per buffer component
- When calling `glColor`, the values specified as arguments are assumed to be in the range 0.0 to 1.0. Example: `glColor3f(1.0, 0.0, 0.0)`
Text rendering

- Sooner or later you may want to render text in your application.
- The course book covers two common types of text primitives:
  - Stroke fonts
  - Raster fonts
- There are also textured fonts, where each character is represented as a textured quad. This technique is extensively used in practice and is implemented in most 3D rendering or game engines.
Text rendering cont.

- OpenGL does not provide text primitives, since they can be created from other graphics primitives.
- GLUT provides a few stroke and raster font sets.
- **FTGL** is an easy to use C++ library for OpenGL text rendering. It provides stroke, raster, and textured font sets.
- Overall, efficient rendering of high-quality, antialiased text that scales well is a surprisingly difficult task. There is usually a trade-off between visual quality and performance.
- One of the current state-of-the-art methods for text rendering is to use signed distance fields, as proposed in this white-paper by Valve.
OpenGL and object-oriented programming

- OpenGL is callable from C++, but is not object-oriented.
- Nothing prevents you from wrapping OpenGL calls inside C++ classes (common in practice), but keep in mind that OpenGL relies on global state and that it is easy to introduce undesirable side-effects.
A few words about modern shader-based OpenGL programming

- The graphics pipeline you have seen in Lecture 1 and in the course book has been around since 1994 and is known as the **fixed function pipeline**.

- Since OpenGL 3.1, the fixed function pipeline has been eliminated and replaced with the **programmable pipeline**, which takes full advantage of modern programmable GPUs and uses so-called **shader programs** to perform most of the vertex and fragment processing.

- Compared with classical fixed function OpenGL, shader-based OpenGL programming allows for faster rendering and more sophisticated rendering techniques, but is a bit more complicated to get started with.

- I will not go into more details here, but will instead cover the topic in Lecture 10 (and perhaps also in Lecture 12).
Color systems

A spinning RGB color cube. Try to implement it yourself!
The RGB/CMY color space

- **RGB** *(Red Green Blue)*: for additive color mixing, e.g., on a computer display.
- **CMY** *(Cyan Magenta Yellow)*: for subtractive color mixing, e.g., in printing.

\[ [C, M, Y] = [1, 1, 1] - [R, G, B] \]

Note: Color values in OpenGL range from 0-1, not 0-255
Color mixing

R + G + B = white (additive)

C + M + Y = black (subtractive)
The HSV/HSL color spaces

- **Hue** (angle)
- **Saturation** (radius)
- **Value, Lightness** (height)

- The HSV and HSL color spaces decouple intensity from color information.

CC image courtesy of SharkD
RGB vs. HSV

RGB

HSV

H

S

V
Choosing color space

- OpenGL assumes that colors are represented as RGB (`glColor3f`) or RGBA (`glColor4f`).

- Color spaces that decouple intensity from color are useful in some situations, e.g., when applying image-based postprocessing effects.

- You can use HSV and other non-RGB color spaces internally in your application program, as long as you convert the resulting colors to RGB before rendering.

- Regardless of which color space is used in the application, RGB is the norm in modern displays.
The human visual system

- The human eye has two types of light sensors: **rods** and **cones**.
- Rods sense luminance, or "brightness", but not color. They have slower response time than cones but are about 100 times more sensitive to light.
- There are three types of cones:
  - S, blue-sensitive
  - M, green-sensitive
  - L, red-sensitive
The mantis shrimp

- The eyes of the mantis shrimp has 12 types of color receptors and can perceive both ultraviolet and infrared light.
- Each eye has depth perception and trinocular vision.
- The powerful claws can deliver punches with a force of 1500 N...
True-color frame buffer

- Store RGB values directly in the frame buffer
- Each pixel requires at least 3 bytes => $2^{24}$ (16 millions) simultaneous colors
- De facto standard in modern graphics systems
Indexed-color frame buffer

- Store index into a color map in the frame buffer
- Each pixel requires at least 1 byte => $2^8$ (256) simultaneous colors
- Used in early (ancient) graphics systems mainly because of its lower memory requirements compared with RGB color. Replaced by RGB color in modern graphics systems (since the extra memory cost of RGB color is no longer an issue).
- The concept of indexed color is still used for pseudocoloring of, e.g., grayscale images
Earthquake visualization that maps the strength of quakes (here represented as spheres) onto color.
Gamma correction

- Most displays have non-linear intensity scales. The most common correction method is called gamma correction:

  \[ y = x^{\frac{1}{\gamma}} \quad \text{if} \quad 0 \leq x \leq 1 \]

where \( x \) is the input intensity and \( y \) is the output intensity.
Color blending

Original image 1

Original image 2
Additive and subtractive blending

**Additive blending**: \( C = A + B \)
Used for, e.g., combining light

**Subtractive blending**: \( C = A - (1 - B) \)
Used for, e.g., filter effects
Average and multiplicative blending

**Average blending:** $C = \frac{(A + B)}{2}$
Used for, e.g., anti-aliasing

**Multiplicative blending:** $C = A \times B$
Used for, e.g., combining light and matter
About the first lab

- The first lab session will be held on Monday 25 March, 13:15-17:00, in 1312D & 1313D.
- Your task will be to implement a turtle graphics API :)
Instructions and source code

- Instructions and source code for the assignment can be found at Studentportalen.
How you can prepare yourself for the lab

- Find out what graphics card your computer (or the computers in the PC lab) uses and which OpenGL version it supports.
- Install OpenGL (consult Google for instructions).
- Set up a suitable development environment. Suggestions:
  - Windows: Visual Studio + CMake
  - Linux: Vim/Emacs + CMake + Make + GCC/G++
- Try to compile and run one of the OpenGL examples from the course book (to verify that everything is set up correctly).