# Project 1 – Interactive 2D Geometry

*Due*: September 26th (11:59pm)

**Overview:** This assignment is intended to be a hands-on introduction to real-time graphics programming. The goal is to develop a simple application that uses many of the basic features of the OpenGL API, and supports user interaction with what is drawn on the screen. The application will draw a single, textured square in the center of a window, and should allow a user to resize, move and rotate this square.

**Preparation:** Homework 1 will take you through many of the basic geometry operations you need to use in this project. I would suggest looking at that before doing anything here beyond the basic feature of just getting to code to compile.

**Basic Features (50 points):**

- Installing all libraries and compiling the starter code.
- Drawing a square that moves when the user clicks and drags their mouse.

**Required Features (45 points):**

(5) Translating only when the user drags on the square's interior (but not on the edge or outside the square)
(10) Scaling the square when the user drags on the corners
(15) Rotating the square when the user drags on the edges
(5) Texture the square with an image loaded from a PPM file
(5) Significantly brighten the image on the square (without distorting the colors)
(5) Pressing 'r', resets the square's position, scale, and orientation

**Extra Features (up to 15 points):**

(5) Changing something new with the shape's visuals when the user presses the keyboard (e.g., changing textures, brightness, background, etc.)
(5) Supporting a textured triangle (e.g., toggle between triangle and square)
(5) On-screen graphical indicator of what motion is being performed.
(5) Animating the square (without using any per-frame user input)
(5) Multiple squares that can be controlled simultaneously
(5) Maintaining the square's shape when the window aspect ratio changes

**Submission Details:**
Submission should be in the form of a link to a webpage. The webpage must contain:
 - An image of your program running
 - A short description (~1 paragraph) of any difficulties you encountered
 - A zip file with all the code needed to compile your project along with a working executable (note which OS it is for).

All material on the webpage must be shared with course staff. If we have to request permission, expect to lose points!

Points over 100 (up to 110 total) will count as extra credit.

**Notes & Tips:**

- For full points, you need to get smooth user interactions. Don't have the square jump or suddenly rotate when it's clicked on.

- This project is intended to be a mini-tour of graphics programming. As such, there are lots of different aspects of it to work on. If you get stuck on one area give up on it (but, only for a while!) and move to some other aspect of the project.

- It can be a pain to get the libraries working and the sample code compiling. Start early with this aspect. Remember, you get over half of the points on this assignment just for getting the starter code to compile!

**-** I know it might have been a while since you last did file input and output in C++. Here is a simple example that opens a PPM file, and prints out all the information:

```
ifstream ppmFile;
ppmFile.open("test.ppm");
string ppmType;
int w, h, maxVal, red, green, blue;
ppmFile >> ppmType;
ppmFile >> w >> h >> maxVal;
printf("PPM Type: %s. Image size %d x %d. Max Val: %d.\n",
       ppmType.c_str(), w, h, maxVal);
int pixelNum = 0;
while (ppmFile >> red >> green >> blue){
       printf("Pixel# %5d, R: %d  G: %d  B: %d \n",
             pixelNum, red, green, blue);
       pixelNum++;
}
```

This is just one example. It's important to remember that there are many ways of doing file IO in C++, feel free to use a different approach if you are more comfortable with that.

-I've marked several parts of the sample code as "`TODO: TEST`". These mark something interesting to try out, to build a better understanding of how the OpenGL codes works. Try these out!

**-**You can convert your own images to PPMs, just make sure they use the same format we discussed in class (or make sure you update your parser to handle the other formats!). In Linux and Mac OS X I recommend installing ImageMagick, you can then covert any image to a PPM using the following command:

```
> convert -compress none example.jpg example.ppm
```

**Getting Started:**
You will need the starter code, which is available from the course webpage. To complete this assignment, you will also need to install development libraries for SDL3 and OpenGL, and should update all your relevant graphics and display drivers.

Note--these instructions may have issues with any given machine. Please be sure to check the class forums or email the course staff if you get stuck.

**Installing SDL3, OpenGL:**

OS X:
1. Download Xcode to get GCC/LLVM and other development tools

2. Download a recent SDL3 .dmg files under the "Assets" section of this link:
https://github.com/libsdl-org/SDL/releases/latest
(choose SDL3-3.2.20.dmg or later)

3. This .dmg will include a file called *SDL3.xcframework* this is a folder. Open it and find a subfolder called *macos-arm64_x86_64*. Within this folder there is a folder called *SDL3.framework*. Copy the folder SDL3.framework to the directory: `~/Library/Frameworks`.
Note: I placed the folder *SDL3.framework* in `~/Library/Frameworks`, you can put it on other directories, just make sure it matches the `–F` parameter and that your `–rpath` matches when you compile as shown below.

4. OpenGL should already be installed as part of the OS. However, please make sure you have updated to the latest OS release that your computer supports. See:
https://www.khronos.org/opengl/wiki/Getting_Started

Linux:
1. GCC is likely already installed.

2. You will need to install the SDL3 development libraries package: `libsdl3-dev`
e.g., `sudo apt-get install libsdl3-dev`

3. Parts of OpenGL likely come with your distribution, but you'll often need to installed additional development packages (e.g., with apt-get). Typically:
libgl1-mesa-dev and mesa-common-dev.
e.g., `sudo apt-get install libgl1-mesa-dev mesa-common-dev`

Windows:
1. You should install MS Visual Studio.

2. Download and install the SDL2 development libraries from:
https://github.com/libsdl-org/SDL/releases/latest. Be sure to get ***the development libraries*** with *-devel-* in the name, which will include the header files and the .lib

files. For example: https://github.com/libsdl-org/SDL/releases/download/release-3.2.20/SDL3-devel-3.2.20-VC.zip

3. OpenGL is mostly included in windows, but be sure to install the latest drivers for your graphics card (see https://www.khronos.org/opengl/wiki/Getting_Started).

**Compiling and running the starter code:**
On OSX or Linux, be sure to specify that you need to link against SDL3 and OpenGL. You need to include both the code for Square.cpp *and the code for glad.c* when compiling.

OS x:
```
clang++ -Wall square.cpp -xc glad/glad.c -F ~/Library/Frameworks
-framework SDL3 -Wl,-rpath,$HOME/Library/Frameworks -o square
```

Linux:
```
g++ Square.cpp glad/glad.c -lSDL3 -lSDL3main -lGL -ldl
  -I/usr/include/SDL3/ -o square
```

Your actual command line call may vary. Typing `locate SDL.h` will tell you where the SDL.h file is located, and you can update the directory after –I to match.

*Getting comfortable with Make (or CMake) can help support you as you move to compiling large projects with multiple files, libraries and dependencies.*

Windows:
Be sure to include both `SDL3` and `SDL3main` libraries when linking. Be sure to include both `Square.cpp` and `glad/glad.c` when compiling.