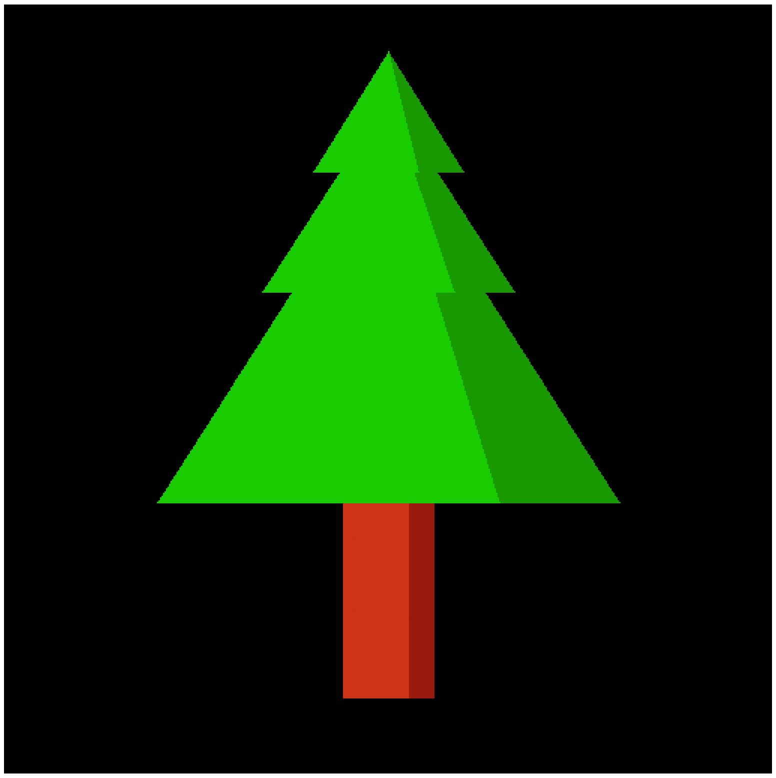
Assignment 2: OpenGL Lite

Covers Lectures 3 and 4

CS 148 Autumn 2013-2014

Due Date:



Introduction. The goal of this assignment is to learn more about the inner workings of OpenGL by using a small subset of OpenGL, which we call SGL (Simple Graphics Library). To help you with this task, you will be using libST. This library will help you primarily by abstracting the drawing primitives: vertices, triangles, transforms, etc. We will provide you with a sample code done by the TA. You will modify the code to create some art of your own.

OpenGL is a big library, so for this assignment we stripped it down to its core components. From a birds eye view, SGL will allow you to draw 2D triangle strips, and specify their color, rotation, translation, and scale. The sample code contains three primary files which you may want to modify: `main.cpp`, `sgl.h`, and `sgl.cpp`.

Sample Code. The sample code contains implementation of triangle strips and different types of transformation implemented by the TA. You will need to build libST first before working on SGL. We have included all of the build files (Visual Studio project, Xcode project, and linux Makefile) in this single download. Just use the appropriate one for your system.

1. Build the libST toolkit. libST source code is located in the `/libst` subdirectory. In this directory there is a `Makefile` for building with `g++`. A Solution file for Visual Studio 2010 is located in the `/libst/vc2008` directory. A Visual Studio build of the library should result in the creation of `/libst/lib/{BuildConfiguration}/libst.lib` (If you are using `g++`, building the library will produce `/libst/lib/libst.a`). If using Visual Studio be sure to build libST for both Debug and Release configurations.
   (a) libST includes a number of headers and source files. Take time to familiarize yourself with the library.
   (b) **Make sure to rebuild libST after you are done modifying it.**
2. Build the assignment2 application. We have set up the starter kit so that building the assignment2 application will link to the libST library automatically (as long as the directories are not moved around). The starter code for assignment 2 is located in the directory `/assignment2`. This folder has the following files:
   (a) `libsgl/sgl.cpp and libsgl/sgl.h -` These files declare the SGL library and are the main files you may want to edit, including implementation of triangle strips and different types of transformation.
   (b) `assignment2/main.cpp -` This file handles most of the GLUT interaction; this is where you will place the code that drawing your picture.

OS X: When building and installing libST, make sure to install the dependent libraries (`libpng`, `jpeg`, `freetype`) first if you don't have them already. To install them, first install MacPorts and issue the following commands in the Terminal application:

```
sudo port install libpng
sudo port install jpeg
sudo port install freetype
```

For more information, **consult the readme file accompanying the libST XCode project**.

Linux: You will also need to install the following libraries if they are not already on your system: `libpng-dev`, `libjpeg-dev`, `libftgl-dev`, and `libglew-dev`. Use your package manager to find and install these libraries. If your OS is Ubuntu, you can use the following command on the terminal:

```
sudo apt-get install libpng-dev libjpeg-dev libftgl-dev libglew-dev
```

iOS and Tizen: You will need to cross-compile libST, libSGL and all associated dependencies on the mobile platform. Note that libST will need to modified to use OpenGL ES instead of OpenGL.

Android: One option for Android is to cross compile libST, libSGL, and all associated dependencies and follow the approach described above. Another option is to use the libraries provided by Android for File I/O and write your own vector and point classes instead of using the ones in libST. You can base your vector and point classes off the classes in `javax.vecmath` which are only available on the desktop version of Java and not available in the Android Dalvik VM.

Implementation.

Part 1. libST Before you start implementing, take a look at the libST library. You may also find some other classes that are useful for your assignment. The following is a brief overview.

- **STPoint3** - `STPoint3` implements basic functionality and operators for 3D points.

- **STVector3** - `STVector3` implements basic functionality and operators for 3D vectors.
- **STImage** - `STImage` handles the loading, writing, and copying of images.

Part 2. SGL

- **Triangle rasterization.** For this assignment you will need to draw triangle strips, and it will all be in 2D. We have implemented triangle rasterization (including handling singularities etc.) for you. Please refer to Wikipedia if you are unsure about how triangle strips work. When specifying triangle strips the first three vertices between `sglBeginTriangles()` and `sglEnd()` will form the first triangle. For each subsequent vertex a new triangle will be specified by taking that vertex and the last two on the list. You will store these in some sort of vertex buffer that you can then use to draw triangles once `sglEnd()` is called. You will want to apply any transforms to these vertices (more below in the transform stack section). The lecture slides describe a general algorithm you can follow. Remember that when rasterizing the triangle you will want to compute its bounding box first, and then test which points in that box are inside the triangle. Once you know which pixels correspond to points in the triangle you can color it appropriately.

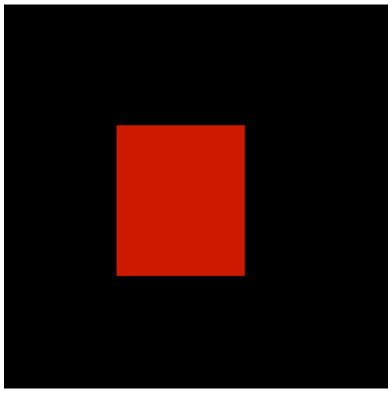
To give you a general idea of how SGL's syntax compares to OpenGL's syntax, here are code snippets that draw the same red square.

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glBegin(GL_TRIANGLE_STRIP);
    glColor3f(.8f, .1f, 0.f);
    glVertex2f(150.f, 150.f);
    glVertex2f(320.f, 150.f);
    glVertex2f(150.f, 350.f);
    glVertex2f(320.f, 350.f);
glEnd();

sglLoadIdentity();
sglBeginTriangles();
    sglColor(.8f, .1f, 0.f);
    sglVertex(150.f, 150.f);
    sglVertex(320.f, 150.f);
    sglVertex(150.f, 350.f);
    sglVertex(320.f, 350.f);
sglEnd();
buff->Draw();
```

Please note that the call to `buff->Draw()` *is not part of SGL or OpenGL.* `buff` *is just an* `STImage` *we are using for this assignment, and* `buff->Draw()` *simply allows you to draw the image onto an OpenGL buffer. This call is already included in the display function in the starter code so you don't need to call it yourself.*

- **Color interpolation.** In OpenGL each vertex stores more than its location in 3D space. You can store color information, texture coordinates, normals among other attributes in each vertex. In SGL you will only need to store the color of each vertex, which you will then use when rasterizing your triangle. Color is one of the many attributes OpenGL stores as part of its state. What that means is that after specifying a color all following vertices will be that color until its changed one more time. You will need to interpolate between the colors of each vertex to determine the final color of points that lie inside the triangle. One way to do this, as described in class, is to calculate the barycentric coordinates of each point inside the triangle, and use those weights to determine what percentage of each vertex's color that pixel will get.



- **Transformation stack.** We have also implemented for you functions to allow for rotation, scaling and translation of the triangle strips. Each of these operations is performed by multiplying the vertex's position by a special matrix. Remember that this transform matrix is part of OpenGL's state, meaning that after modifying the current matrix, all following vertices will be affected by it. If we dont want those transforms to take effect anymore we set the current matrix to the identity matrix. For this assignment the four functions that will affect your current transformation are `sglScale()`, `sglRotate()`, `sglTranslate()`, and `sglLoadIdentity()`. OpenGL facilitates setting complex transforms on the scenes 3D objects by providing a stack (as in data structure) of matrices. By pushing and popping transforms onto the stack you can create hierarchical relationships between objects in the scene. In SGL we create a stack of transforms. Unlike OpenGL it does not have to be of a certain size, and you are welcome to use the C++ standard library to help you with this assignment.

- Create your own work of art. The purpose of drawing machines is to draw pretty pictures. This is the main part of the assignment. You should draw some pretty pictures using triangle strips and different types of transformation. For example, you can modify the vertex positions of the triangle strips; you can draw more triangle strips; you can use different types of transformation to create a picture out of the triangle strips.

- You will be expected to know the answers to these questions when you present your project to a CA for grading.

  1. In class we discussed that OpenGL works primarily on points, lines and triangles. What's the advantage of using triangles as the main polygon from which to build objects as opposed to others like quadrilaterals?
  2. Why do we compute the triangles' bounding boxes during the rasterization step?
  3. Why do we use homogeneous coordinates in OpenGL and SGL?
  4. What are the advantages of having OpenGL use a state machine when drawing shapes?
  5. What is the advantage of implementing a matrix stack rather than having a single transform matrix?
  6. Suppose we have two squares, one centered at `(10,0)` and the other at `(-10,0)`. We want to animate them so that they each rotate around their center and also rotate around the origin (think of spinning planets rotating around the sun). Describe the transform stack that you would use to create such an animation.

Grading This assignment will be graded according to the following rubric.

- + – Exceeds the requirements via one or more artistic/technical contributions
- ✓ – Meets all of the requirements

- − − Does not meet the requirements but still produces a drawing.
- 0 – The submitted solution does not produce a drawing.

FAQs

1. If libST fails to compile in Xcode, with errors complaining about `"vnsprintf"` not being in `std`, try to do the following:
   - Go to project → Edit Project Settings.
   - Change configurations to "All Configurations" and click on the "Build" tab.
   - Find the setting "Header Search Paths" (you can use the search box or scroll down to "Search Paths".
   - Double click it. Make sure "recursive" is not checked.
   - If from here, you get compile errors about not finding a `freetype` file, change the search path for `freetype2` to: "`/opt/local/include/freetype2`".
2. If you cannot install `freetype` using MacPorts, try doing:

   ```
   sudo port uninstall zlib and sudo port install zlib
   ```
3. With Xcode, if you've verified that you have `libpng`, `libjpeg`, etc. installed and still get linking errors with a warning "`libjpeg.dylib files is not of required architecture`", please make the following changes to libST, libSGL, and assignment2:
   - Go to project → Edit Project Settings.
   - Change configurations to "All Configurations" and click on the "Build" tab.
   - Change architectures from "`$(NATIVE_ARCH)`" to "Native Architecture of the Build Machine".
   - Recompile libST and then assignment 2.
4. If libST fails to compile, try adding standard headers like `<stdio.h>`.