CS 4053/5053
Homework 02 – Drawing and Animating Scenes with OpenGL
Due Tuesday 2024.02.20 at 11:00pm.

All homework assignments are individual efforts, and must be completed entirely on your own.

In this assignment you will learn how to build and animate a moderately complicated 2-D scene using JOGL. Specifically, you will use a mix of OpenGL drawing primitives and rendering options to create an animated scene inspired by a hand-drawn reference picture.

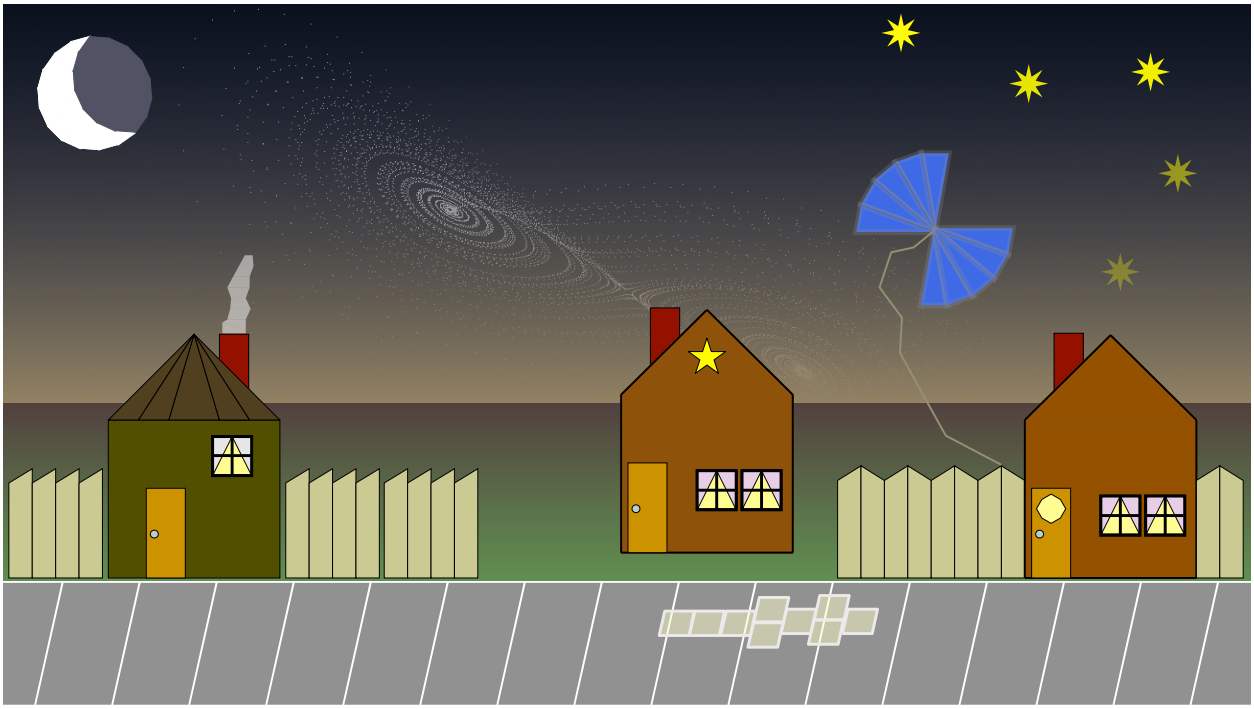
Getting Started

Reproducing the structure and style of artistic assets is a common task in computer graphics, especially in large game projects. Consider the reference picture below. It was drawn in Keynote '09 for Mac OS X. The scene contains a variety of graphical elements composed of points, lines, and polygons. The polygons have a mix of sides and shapes, and all are filled with either a solid color or a linear gradient. Some lines and edges are thicker than others. A few elements are even translucent. The elements were all created, positioned, and styled using common drawing features, except for the galaxy, which is a stretched screenshot of our OpenGL Lorenz program.

Exploring an Example Scene

Study how a moderately complicated polygon-drawing program can be written using JOGL. The `polygons` program displays a street scene with a few houses, fences, and a kite. It reproduces some of the structure and style of the reference picture. Examine the code in `Polygons.java` to see how the graphical elements are constructed, and also animated by varying line thickness.

Notice especially how scene construction can be organized as clear, reusable chunks of code. Each chunk either uses `GL_POLYGON`, `GL_TRIANGLE_FAN`, or `GL_QUAD_STRIP` to fill an instance of some type of graphical element, or a `GL_LINE_LOOP` to draw the element's edges.



Creating a New Scene

Imagine a new scene like the one in the reference picture, then write code to view it. Feel free to be creative with your scene design, but make sure to include all of the following features:

- The sky and ground, drawn with gradients. *(Hint: You can vary vertex colors to fill any polygon with a gradient.)*
- A background galaxy, generated with the Lorenz attractor, Tinkerbell map, or similar function. Your galaxy code and the graphical result must be clearly different from my examples. *(Hint: You can use multiple projections to render layers in a scene. Notice how setProjection() differs in Lorenz.java and Polygons.java.)*
- At least two houses, all clearly different from the `polygons` houses and from each other.
- Four of these house features: a divided roof, a window with shades, a door with doorknob, a door with window, a window with thicker frame, a smokestack with smoke, a five-pointed star.
- A pair of fences having at least six boards each, one jagged and one alternating.

Also include any two of the following in your scene:

- A novel (but recognizable) house feature of your own devising.
- A kite with a string that is tied to a house feature or the top of a fence board. The geometry of the kite itself must be visibly different from the one drawn by the starter code.
- A sidewalk with a feature of your choosing (hopscotch court, manhole cover, storm drain, etc.)
- A flagpole with a flag and rope.
- A fire hydrant (water spray/drip/pool/etc. not required).
- A crescent moon with a shadow.
- Five or more 8-point stars that appear faded the closer they are to the horizon. *(Hint: Clever use of an unexpected drawing mode makes it much easier to fill the stars correctly.)*

Finally, animate your scene. Pick any two of your features from either list above. Vary each one over time in terms of location, size, geometry, and/or color. The behaviors of both features must make sense to a typical viewer, but can be imaginative. Set up each animation to progress at a watchable rate. The animations can have phases or repeat if you like. The animations must be non-trivial, clearly different in character, and not simply reuse the example line thickness and polygon fill variations included in the starter code.

For this assignment, navigate to the `edu.ou.cs.cg.assignment.homework02` package and modify `Application.java`. It's a copy of `Polygons.java` to help you get started. Use as many private methods as you like to appropriately modularize your code, but don't create any new Java files (this time). Document your code thoroughly. The corresponding program is `hw02`.

Turning It In

Turn in a complete, cleaned, renamed, zipped **COPY** of your **ENTIRE** `homework02` directory:

- Never delete `About` or `Results`! Preserve all file structure and contents of the assignment's original zip download, except for any modifications and additions specified in the instructions.
- Take a screenshot of your application window when it's in an interesting graphical state.
- Put the screenshot in the `Results` directory as `snapshot.png` or `snapshot.jpg`.
- Go into the `ou-cs-cg` directory.
  - Make sure it contains all of the code modifications and additions that you wish to submit.
  - Run `gradlew clean` (on the command line). This should remove the `build` directory, reducing the size of your submission. We will clean and rebuild when we grade regardless.

  - If you used an IDE, remove any IDE-specific leftovers (such as the Eclipse `bin` directory).
- Append your 4x4 to the `homework02` directory; mine would be `homework02-weav8417`.
- Zip your entire renamed `homework02-xxxx####` directory.
- Submit your zip file to the Homework02 assignment in Canvas.

These steps will make your submissions smaller and neater, which speeds up grading a lot.

You will be scored on: (1) how many of the feature and animation requirements that you satisfy; (2) the overall appeal of your scene; and (3) the clarity and appropriateness of your code. Creativity is encouraged; meet the requirements, but beyond that extend and vary the design as much as you like to make the look and feel of the scene your own. ***It's fine (and expected) for the locations, sizes, and colors of features to differ from the reference picture!***