Assignment 2 COMP 557 Winter 2015

Posted: Wednesday, Feb. 18

Due: Thursday, March 5 at 23:59

General Information

- This assignment is worth 12 % of your final course grade.
- The assignment code was written by Fahim Mannan, and any design questions may be directed to him. Otherwise, please direct your questions to Charles Bouchard and Shayan Rezvankhah who will handle the office hours and grading. Office hours will be posted on the public web page.
- Use the myCourses discussion board (Assignments/A2) for clarification questions.
- Do not change any of the given code. Add code only where instructed.
- Submit the entire directory as a `zip` or `tar` file to the myCourses Assignment/A2 folder. The file should be named `FirstnameLastname.zip` (or `.tar`) and should unpack into a directory with that same name.

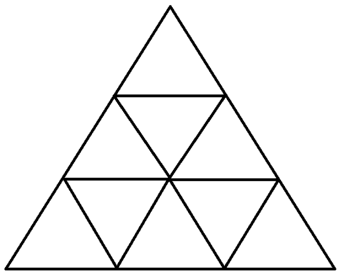
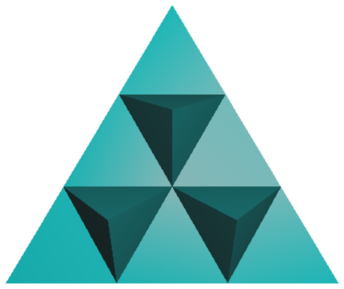
  If you have any issues that you wish the TAs to be aware of when grading, then include them as a separate comment with your submission.
- **Late assignments** will be accepted up to only 3 days late and will be penalized by *1 point per day on your final course grade.* If you submit one minute late, this is equivalent to submitting 23 hours and 59 minutes late, etc.

  Note that the assignment has 120 points, so the late penalty is 10 points per day.

Questions

1. **Koch Pyramid (20 points)**

   The Koch pyramid is defined as follows. Start with an equilateral triangle, and partition it into nine subtriangles (see below left). Delete three of these sub-triangles and replace each of them with three faces of a pyramid (see below right). Each pyramid is a regular tetrahedron (Platonic solid) without the bottom.

   Implement the `drawKochPyramid()` method of the `KochPyramid` class. You may define helper methods if you wish. You may also change the lighting and material if you wish to improve the appearance.

   Notes: (1) The starter code draws a shiny sphere to help you visualize the lighting. (2) The cosine of the dihedral angle (elevation of the side) of a regular tetrahedron is $\frac{1}{3}$, not $\frac{1}{\sqrt{3}}$, *i.e.* you might have assumed incorrectly that the dehedral angle was 60 degrees, i.e. $\cos 60° = \frac{1}{\sqrt{3}}$.

2. **L-system (30 points)**

   An L-system parser is provided in `LSystem.py` . The parser recognizes the specification given in the table below.

   All implementation of the L-system parsing and rendering are in the class `LSystem`. The L-system grammar is specified in a python dictionary of the following form:

   - `init` : The initial string. Any sequence of characters can be used.(**required**)
   - `rule` : The production rule with the left and right sides separated by `->`. Multiple production rules are separated by a semi-colon. Any character (case-sensitive) can be used for the production rule. (**required**)
   - `depth`: The number of times production rules will be applied (**required**)
   - `angle`: The default rotation angle for the rotation commands. Default angle 0. (**optional**)
   - `growth_factor`: The amount by which the L-System will be scaled ($\texttt{growth\_factor}^{\texttt{depth}}$). Default factor is 0.5. (**optional**)
   - `init_angle_axis`: Initial orientation of the model expressed in angle-axis form. (**optional**)

- `init_translate`: Initial translation to be applied. (**optional**)
- `XZScale`: X and Z axis scale. Default scale factor is 1. (**optional**)
- `YScale`: Y axis scale. Default scale factor is 1. (**optional**)

Table 1: Table. Interpretation of L-system Alphabet

F	Go forward (x direction) and draw a line
f	Go forward (x direction) but do not draw a line
L	Draw a leaf e.g. a green triangle
B	Draw a branch e.g. a brown cylinder
W	Scale up in the x and z direction (see `self.XZScale`)
w	Scale down in the x and z direction
S	Scale up in the y direction (see `self.YScale`)
s	Scale down in the y direction
P	"Pitch": Rotate counter-clockwise about the x-axis
p	"Pitch": Rotate clockwise about the x-axis
Y	"Yaw": Rotate counter-clockwise about the y-axis
y	"Yaw": Rotate clockwise about the y-axis
R / '+'	"Roll": Rotate counter-clockwise about the z-axis
r / '-'	"Roll": Rotate clockwise about the z-axis
[	Push the current transformations (matrix and attributes)
]	Pop the current transformations

How does it work? To create an L-System, the user has to provide the L-system grammar along with some parameters such as the number of levels/depth, angle of rotation, growth factor, initial orientation using axis angle specification. An example is as follows:

```
Plant2D = {'init': 'F',
'rule' : 'F->FF-[-F+F+F]+[+F-F-F]',
'depth': 3, 'angle': 22.5, 'growth_factor': .5,
'init_angle_axis': {'angle': 90, 'axis':[0, 0, 1]},
}
```

The L-system can be instantiated by calling `LSystem(Plant2D)`.

The sequence of operations is as follows. Once an L-system is instantiated, the `set_grammar()` function gets called which is responsible for parsing the specification and expanding the initial string to the final string. The final string is stored in a variable called `production`. Every time the L-system needs to be drawn, the `draw_scene()` function is called which will go over the final string and treat it as a sequence of draw commands. It will execute those commands by calling `exec_draw_cmd()`.

(a) Implement `exec_draw_cmd()`. Your solution will be a large `if-elif` block that calls the appropriate OpenGL commands that correspond to the symbols defined in the table below.

(b) Design 3 different plants. The goal here is to motivate you to get some hands-on experience with L-systems and to experiment with them. For inspiration, see the book "The Algorithm Beauty of Plants" `http://algorithmicbotany.org/`.

Your plants should be added in the `main` method. See the instructions there.

3. **Hermite Curve (20 pts)**

`HermiteCurve.py` can be used to define a Hermite curve (or "spline"). See the comments within the file for details. Also see the unit test module `HermiteCurveTest.py`.

(a) Implement `add_point()` that takes a datapoint and builds the coefficient matrix for interpolation.

(b) Implement `evaluate_curve_segment()` which takes a segment number and $t \in [0, 1]$ and returns a tuple of point and tangent.

4. **Bicubic Patch (30 pts)**

The module `BicubicPatch.py` can be used to define a single bicubic patch. Unit test code `BicubicPatchTest.py` is also provided.

(a) Write the code for constructing and evaluating a bicubic patch, so that for a given (s, t), you can compute $(X(s, t), Y(s, t), Z(s, t))$ as well as partial derivatives in the s and t directions. This requires adding the missing code of the `evaluate()` method.

(b) Implement the `set_color_from_XYZ()` method which chooses the color of a point on a bicubic patch. Since we do not have lighting in the garden scene (Q5), we need to distinguish points based on this color. The default method given to you makes the RGB components vary with xyz, respectively. For the garden scene, you should use a more appropriate color scheme which would be to make all surface points greenish *e.g.* go from dark green to light green (or white) as "height" of the surface increases.

(c) Implement the `draw_scene()` method for drawing a bicubic patch. This method should create a triangulation or quadrangulation and should color the points appropriately. It is not necessary to create surface normals here since lighting is not being used.

5. **Garden Scene (20 pts)**

   Combine Q2 - Q4 in a single garden scene. The garden should contain several elements:

   (a) The ground must be a *regular terrain* that is defined by a bicubic patch. The ground must be curved in both the x and y directions, and this curvature should be visually noticeable. A sloped plane is not allowed.

   (b) A tiled pathway. The tiles must made up of "thick" quads, just like tiles in a real garden. (Scaled cubes are fine.) The tiles must be oriented so that the normal of the upper surface is aligned with the local normal of the bicubic patch.

   (c) The L-system plants from Q2.

User Interface

All windows provide the same interactions as in Assignment 1 and more. Note that for keyboard interactions, the characters are case-insensitive. Mouse buttons are represented as L/M/R BD or Left/Middle (or Scroll wheel)/Right mouse Button Down.

keyboard	mouse	effect
'a'/'d'	RBD left/right	Translate camera left/right of lookat
'w'/'s'	RBD up/down	Translate camera forward/backward along the lookat
'r'/'f'		Translate camera up/down along the up vector
'z'/'x'		Rotate ("roll") CCW/CW about camera z axis
←/ →	LBD left/right	Rotate ("yaw") left/right about the camera y axis
↑/ ↓	LBD up/down	Rotate ("pitch") up/down about the camera x axis
Ctrl + (←/ →)	MBD left/right	Arcball rotate left/right
Ctrl + (↑/ ↓)	MBD up/down	Arcball rotate up/down
	scroll up/down	Move model farther/closer
The actions below are for `Garden Scene` window only. That window must be active.		
'p'		Add a point and a tangent for fly through camera.
'g'		Animate the flythrough. Camera follows curve.
'c'		Clear the flythrough curve.

How is the above implemented? As before, `GLUTWindow` handles all the `glut` based windowing functionalities. In addition to that, now we have a `View` class that handles all the view/camera transformations. The `View` class implements a general camera that can rotate and translate in its local coordinate frame. It can also perform the Arcball/Trackball rotations and zooming (using CTL key) which are very useful! The `FlythroughCamera` class used in `GardenScene.py` extends the `View` class and provides some additional controls ('p', 'g', 'c') for setting up the Hermite curve and fly through animation in Q5. For further details on the implemention, see the `README` file.