CIS*4800 Graphics * Assignment 4/Term Project
Brett Foster, 0239553

Please review the final 3 pages for details of what was accomplished and what the limitations are. The body of this document is provided for reference.

My particular area of interest for the term project is a model for rendering 3D trees using Open GL. The term project will encompass tree generation, and construction. To accomplish this I will be referencing three papers. In increasing level of difficulty these papers are "Real time design and animation of fractal plants and trees", "Modeling the mighty maple", and "Creation and rendering of realistic trees". The latter paper references and discusses the former two papers, however it will only be used as a source of advice towards enhancing previous models and algorithms.

Tree Generation

Tree generation is the method of creating realistic looking (circa 1985) tree structures. Tree generation will be done offline, and later loaded in to an application that will construct the 3D model for the tree. Oppenheimer suggests a fractal based method using parameters such as branching angle, branch-to-parent size, stem taper rates, helical twist, and branches per stem. In addition it used a stochastic model to attempt to remove the deterministic elements of the algorithm.

Basic Algorithm:

```
1.  -- Variable Declarations (defined below)
2.  var helical_twist_angle : ℝ
3.  var stem_stem_ratio     : ℝ
4.  var branch_stem_ratio   : ℝ
5.  var branching_angle     : ℝ
6.  var branch_segments     : ℕ0
7.  -- Transformation matrix for the stem to be multiplied with the active matrix (glMultMatrix)
8.  procedure  transform_stem : Matrix (4x4)
9.  -- Transformation matrix for the branches to be multiplied with the active matrix (glMultMatrix)
10. procedure  transform_branch : Matrix (4x4)
11. -- Writes serialized instructions for rendering a tree to a file.
12. procedure tree ( depth : ℕ0 ) : void
13.    -- At depth 0, load the identity matrix (may be skipped)
14.    if ( depth = 0 )
15.       write ( LOAD_MATRIX, IDENTITY_MATRIX )
16.    write ( MATRIX_PUSH_AND_LOAD )
17.    write ( DRAW )
18.    -- The exit condition for recursion is when the branch is too small
19.    -- to hold any more branches. The true definition may vary based on
20.    -- circumstances.
21.    if ( depth < branch_segments )
22.        -- Generate the next branch segment
23.        -- This gets tricky - The tree is actually generated from
24.        -- the top down. Imagine generating all the segments of the
25.        -- branch first (the stem) then drawing the branches from the top
26.        -- down. The reason for generating branch segments is so that
27.        -- helical twists can be computed for the branches.
28.        Write ( LOAD_MATRIX, transform_stem( ) )
29.        tree ( depth + 1 )
30.        -- Generate branches.
31.        for 0 to branch_segments
32.            write ( LOAD_MATRIX, transform_branch( ) )
33.            tree ( depth + 1 )
34.    write ( MATRIX_POP )
```

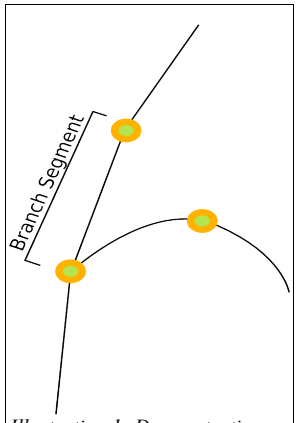

Illustration 1: Demonstrating the branch segment that one call to Tree () will generate. Also demonstrated is the ability of a stem to be curved.

- The constants `MATRIX_PUSH_AND_LOAD`, `MATRIX_POP`, and `DRAW` are commands that serialize the recursive algorithm in terms of OpenGL concepts. (I.e. The OpenGL program rendering the tree reads the commands serially without any recursion.)
- The constant `LOAD_MATRIX` is a command that loads a matrix in to memory. It will be glMultMatrix with the current matrix when the `MATRIX_PUSH_AND_LOAD` command runs.
- The functions `transform_stem`, and `transform_branch` have varying levels of complexity depending on how they are implemented. They will be simple initially, producing simple repetitive trees. A stochastic method will later be applied using standard deviations and randomness to provide variation. (This implies that each parameter will have an associated standard deviation attached to it.)
- The function `transform_stem` returns a matrix based on global parameters:
  1. `helical_twist_angle : ℝ`

The degrees of twist found in the branch/stem segments. Higher values result in more twisting in the resulting tree.

2. `stem_stem_ratio : ℝ`
   The rate at which the stem decreases in size.

- The function `transform_branch` returns a matrix based on global parameters:
  1. `branch_stem_ratio : ℝ`
     The ratio between the branch and the stem size. This parameter sets the size of the branch/stem segments that are part of the child branch.
  2. `branching_angle : ℝ`
     The angle at which branches are made.
- The question "`too small?`" uses the following parameter:
  1. `branch_segments : ℕ₀`
     The total number of branch segments on a branch. The question of being too small is answered yes when the `depth` variable is equal to `branch_segments`.
- Weber proposes a much more flexible method for generating trees with over 40 parameters documented in the appendix of the paper. Although the method is fundamentally different than the proposed fractal method, some parameters may still be introduced as an addition to the algorithm provided here.
- The length of a branch is fixed at 1. It is scaled, and generally manipulated smaller and smaller by transformation matrix.

Tree Construction

To construct the tree in OpenGL the sequence is loaded in to memory and interpreted during the display phase. Initially the display algorithm will simply connect lines. Interpolation code will be added and the branch segments will be interpolated using splines. I will then use polygons to give the tree realistic shape. Finally, I will investigate bump mapping, and the inclusion of a texture.

Fall Back and Enhancements

The fall back position of this project has already been documented. I will start with the simplest parameters and method as specified in Oppenheimer's "Real time design and animation of fractal plants and trees". Enhancements such as the inclusion of new parameters to this algorithm will be made as documented above based on all three papers.

Objectives/Milestones:

1. Basic Tree Algorithms
2. Tree Line Drawing
3. Interpolated Line Drawing
4. Polygon Drawing (4b: add interpolation)
5. Enhanced Parameters for more interesting trees
6. Texturing and Bump Mapping

These objectives were selected because they build on each other allowing for fall back at any point in the process.

Results

The offline portion of the algorithm was adapted and integrated with the online portion to exploit OpenGL's matrix operations. In its place "Tree DNA" (tDNA) is used to specify the trees. Several strands of tDNA are appended together for form a complete definition of a tree that will be rendered consistently every time. (The 'tree' command will generate N data values with a particular mean and standard deviation.) With specialized tools and time it is possible to specify a tree down to its last twig. In testing the results using the `tree` tool were satisfactory. Sometimes `tree` produced trees that looked more like roots than actual trees. The online processor can accept and position multiple trees in one scene.

```
@echo GENERATING tDNA SEQUENCES tree
echo -n > test.dna
@echo DEFAULT_SCALE
./tree 1 50 0 >> test.dna
@echo STEM_DETAIL
./tree 1 5 0 >> test.dna
@echo STEM_RADIUS
./tree 1 0.5 0 >> test.dna
```

```
@echo BRANCHING_SEGMENTS
./tree 1 2  0 >> test.dna
@echo BRANCHING_AROUND
./tree 128 6 3 >> test.dna
@echo STEM_STEM_RATIO
./tree 128 0.95 0.005 >> test.dna
@echo BRANCH_STEM_RATIO
./tree 1 1 0 >> test.dna
@echo BRANCHING_ANGLE_V
./tree 128 30 20 >> test.dna
@echo BRANCHING_ANGLE_H
./tree 128 120 100 >> test.dna
@echo HELICAL_TWIST_X
./tree 128 0 8 >> test.dna
@echo HELICAL_TWIST_Y
./tree 128 0 8 >> test.dna
@echo HELICAL_TWIST_LEN
./tree 128 16 0 >> test.dna
@echo COLOR_RED
./tree 128 0.486 0.1 >> test.dna
@echo COLOR_GREEN
./tree 128 0.23 0.1 >> test.dna
@echo COLOR_BLUE
./tree 128 0.23 0.1 >> test.dna
@echo DRAW_MODE
./tree 1 1 0 >> test.dna
@echo ANIMATION_BRANCHING_ANGLE_V
./tree 1 0 0 >> test.dna
@echo ANIMATION_HELICAL_TWIST_X
./tree 1 0 0 >> test.dna
@echo ANIMATION_HELICAL_TWIST_Y
./tree 1 0 0 >> test.dna
@echo ANIMATION_BRANCHING_ANGLE_H
./tree 1 0 0 >> test.dna
```

(Excerpt from makefile.)

Objectives Checklist:

1. Basic Tree Algorithms
   Completed.
2. Tree Line Drawing
   Completed. Line drawings are still supported for fern-type trees that are better rendered by lines. In addition, a tree can begin solid and transition to line-based drawing for finer details.
3. Interpolated Line Drawing
   Partial. Used helical twist count to provide effect of curvature.
4. Polygon Drawing (4b: add interpolation)
   Completed.
5. Enhanced Parameters for more interesting trees
   Completed. Used techniques to add animated effects. Wind can, in theory, be modeled using these mechanisms. The tree tool, the circular buffer API, and the tree class make it easy to add new parameters as well.
6. Texturing and Bump Mapping
   None.

Over all, this term project was successful. The inclusion of more parameters could be implemented to make more realistic trees. Improvements to the rendering would also assist in the creation of realistic trees.

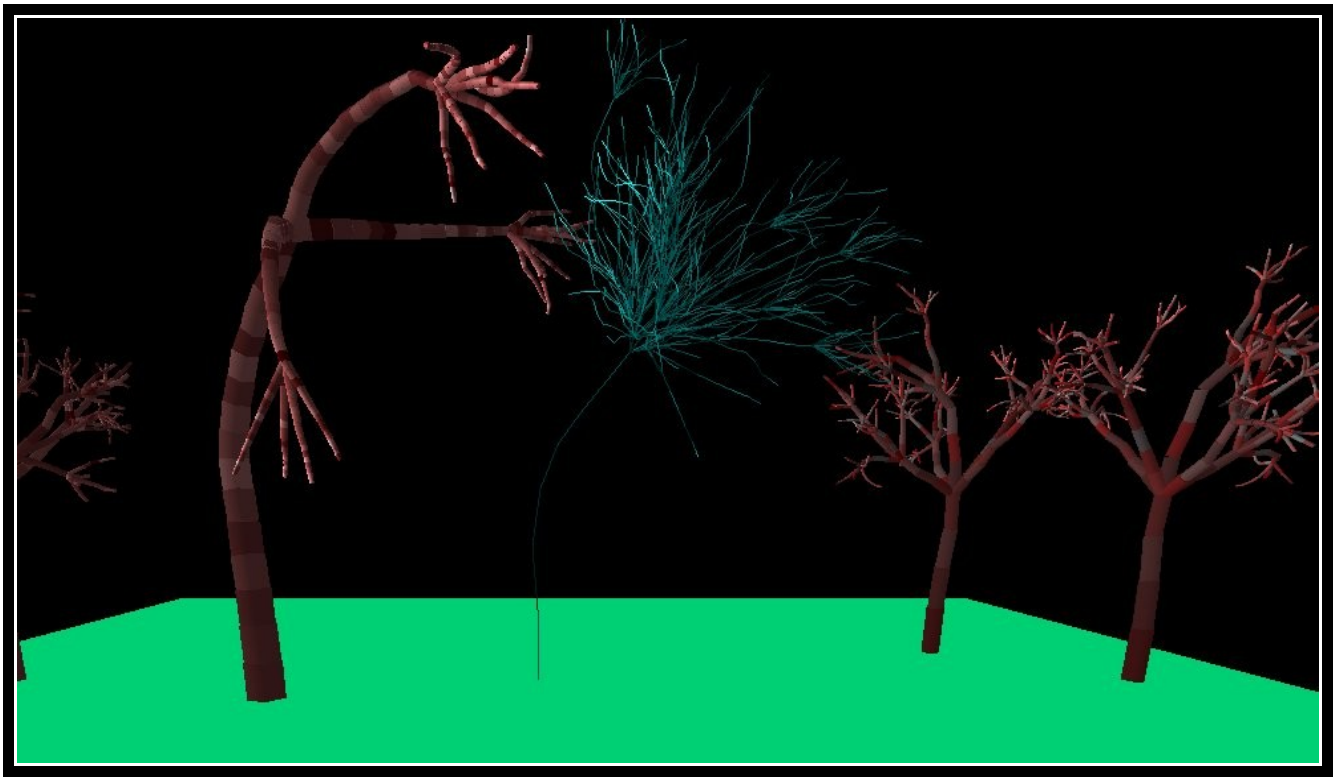

Illustration 2: Tree generation sometimes yielded monster-like creatures! In this animation, the funky looking tree is hitting the funky stringy object.
Command: *cat tree-stringy.dna noanimate.dna noanimate.dna noanimate.dna animate-long-fern.dna weirdo.dna noanimate.dna noanimate.dna noanimate.dna noanimate.dna animate-long.dna tree-3.dna noanimate.dna noanimate.dna noanimate.dna noanimate.dna noanimate.dna tree-2.dna noanimate.dna noanimate.dna noanimate.dna noanimate.dna | ./pongish 0,-50,100,0,160 1,-400,0,0,0 2,800,100,0,0 2,600,300,0,180 3,-800,100,0,0*

Command Reference

```
animate runLength count step
```

Generates a series of floating point values (in tDNA format) from 0 to (count * step) and back to 0 after (2 * count * runLength) values. Generally produces a fading effect with a particular magnitude. Animation effects generated by this tool are fairly random, but periodic making it suitable only to the most basic animations.

```
tree samples mean standard-deviation
```

Similar to the animate tool, however the tree program generates a sequence of random values that has a mean and a standard deviation. (Accuracy of the algorithm producing the random variable has not been verified to match the standard deviation specified.)

```
pongish [treeID,x,y,z,rotation] [treeID,x,y,z,rotation] ... [treeID,x,y,z,rotation] < trees.tdna
```

Loads a series of trees in to memory and positions them for rendering.

TreeID – The tree index based on the order in which trees were loaded.
X,Y,Z – The position of the tree with the basic range of [-1000.0,1000.0]
Rotation – A rotation about the z-axis.

```
make tree-type
```

Generates a tree in file test.dna. Can be loaded with make test.

```
make test
```

Displays the tree in test.dna.

```
make fern-type
```

Generates a line-drawn tree-like structure that is more optimal for drawing fern-type structures. It does not generate objects that looks like a fern, however. I was unable to determine the correct parameters for them.

```
make run
```

Displays the tree in test.dna with no animation effects.

```
make run-precompute
```

Runs the animation in figure 2.

```
make all
```

Compiles important tools!

References

Weber, J. and Penn, J. 1995. Creation and rendering of realistic trees. In *Proceedings of the 22nd Annual Conference on Computer Graphics and interactive Techniques* S. G. Mair and R. Cook, Eds. SIGGRAPH '95. ACM Press, New York, NY, 119-128. DOI= http://doi.acm.org/10.1145/218380.218427

Bloomenthal, J. 1985. Modeling the mighty maple. In *Proceedings of the 12th Annual Conference on Computer Graphics and interactive Techniques* SIGGRAPH '85. ACM Press, New York, NY, 305-311. DOI= http://doi.acm.org/10.1145/325334.325249

Oppenheimer, P. E. 1986. Real time design and animation of fractal plants and trees. In *Proceedings of the 13th Annual Conference on Computer Graphics and interactive Techniques* D. C. Evans and R. J. Athay, Eds. SIGGRAPH '86. ACM Press, New York, NY, 55-64. DOI= http://doi.acm.org/10.1145/15922.15892