1 Administrative Topics

- Office Hour Change: M:1-3:45, T:1-2, 4-5, and I will be available during labs.

- Course Evaluations. important. separate for lecture and lab. we are both invested in the projects. Bruce writes them. We take turns grading.

- For Wed, have some of your simple objects made so you can get our help making more complex objects

2 Project 11

Let’s draw a binary tree. We will require that the binary tree be drawn in exactly one orientation – with the top cube at the top, and the left cubes to the left, and the right cubes to the right. This will make it much more straight-forward to compute the locations of the subtrees.

Let’s use recursion. To do this, we will add a drawBinaryTree method to the Cube class. It will draw the current cube and then make two recursive calls to draw the left and right subtrees. The base case, which is more of a stopping case, will simply return from the function before anything is drawn.
In Section A, we decide to stop drawing cubes when the scale is too small and in Section B, we decide to stop drawing cubes when the scale*distance is too small.

We use the location and scale of the tree as the location and scale of the top cube. To compute the location of the subtrees, we need to use scale, distance, and our knowledge of what (x,y,z) means for a cube. Fortunately, Stephanie has a handy dandy comment saying that if orientation, pitch, and roll are all zero, then (x,y,z) refers to the back upper left corner of the cube.

Here is the code for the method:

```python
def drawBinaryTree( self, xpos, ypos, zpos ):
    # if the scale is too little, then it is time to stop
    if scale < 0.1:
        return

    # draw me
    self.draw( xpos, ypos, zpos=zpos, scale=scale,
                orientation=0, roll=0, pitch=0)

    # draw right subtree
    self.drawBinaryTree( xpos=xpos+self.distance*scale*3.0/4.0,
                         ypos=ypos-self.distance*scale,
                         zpos=zpos+self.distance*scale/4.0,
                         scale=0.5*scale )

    # draw left subtree
    self.drawBinaryTree( xpos=xpos-self.distance*scale/4.0,
                         ypos=ypos-self.distance*scale,
                         zpos=zpos+self.distance*scale/4.0,
                         scale=0.5*scale )
```

Lines 3 and 4 implement the base/stopping case.

Lines 7 and 8 draw the top cube for the tree.

Lines 11 through 14 draw the right subtree. We want the top cube of the right subtree to be half the size of this cube, we want the x-positions to place the cubes a little outside the top cube (i.e. to its right), we want the top of the subtree flush again the bottom of the top cube, and we want it centered in Z.

Lines 17 through 20 repeat the process for the left subtree.
2.1 Adding Color

Let’s lighten the color with depth. To do this, we must set the color as a tuple, rather than by using a string. In class, we assumed it was a tuple, but for the notes, Stephanie is including the appropriate error-checking code.

```python
def drawBinaryTree( self, xpos, ypos, zpos ):
    # if the scale is too little, then it is time to stop
    if scale < 0.1:
        return

    # get the original color, but only if it is a tuple
    if type(self.color) == tuple:
        alterColor = True
        (r0,g0,b0) = self.color
    else:
        alterColor = False

    # draw me
    self.draw( xpos, ypos, zpos=zpos, scale=scale,
                orientation=0, roll=0, pitch=0)

    # if we can, lighten the color for the subtree
    if alterColor:
        # make it lighter, but keep it between 0 and 1
        r = max(min(r0+0.2,1),0)
        g = max(min(g0+0.2,1),0)
        b = max(min(b0+0.2,1),0)
        self.color = (r,g,b)

    # draw right subtree
    self.drawBinaryTree( xpos=xpos+self.distance*scale*3.0/4.0,
                         ypos=ypos-self.distance*scale,
                         zpos=zpos+self.distance*scale/4.0,
                         scale=0.5*scale )

    # draw left subtree
    self.drawBinaryTree( xpos=xpos-self.distance*scale/4.0,
                         ypos=ypos-self.distance*scale,
                         zpos=zpos+self.distance*scale/4.0,
                         scale=0.5*scale )

    # restore the color
    if alterColor:
        self.color = (r0,g0,b0)
```

3
If the color is not a tuple, then nothing happens. If the color is a tuple, then we lighten each channel by 0.2 and set the cube’s color field to the lighter color. In order to ensure all cubes at the same “level” have the same color, we need to restore the color after we have drawn the subtrees.

In lines 20 through 22, we use the max and min functions to enforce lower and upper limits. Note that we use max to enforce the LOWER bound and min to enforce the UPPER bound. This is a bit counter-intuitive at first, so let’s think about it.

Suppose my new color will be too big (e.g. r0+1 is 1.1). Then I want to force it to be 1. So, I want the smaller of 1 and 1.1. Suppose my new color will be fine (e.g. r0+0.1 is 0.9), then I want the smaller of 0.9 and 1. So, min is the right function to use for the upper bound. min(1.1,1) returns 1 and min(0.9,1) returns 0.9.
2.2 Using a Fixed Number of Levels

Let’s switch the stopping criterion so we specify the number of levels in the tree. To do that, we add a parameter to `drawBinaryTree` that indicates how many more levels must be drawn. We decrement it as we recurse, and 0 is our base case.

```python
def drawBinaryTree(self, xpos, ypos, zpos, numLevels):
    # If we have already drawn the lowest level, stop
    if numLevels == 0:
        return

    # get the original color, but only if it is a tuple
    if type(self.color) == tuple:
        alterColor = True
        (r0, g0, b0) = self.color
    else:
        alterColor = False

    # draw me
    self.draw(xpos, ypos, zpos=zpos, scale=scale,
               orientation=0, roll=0, pitch=0)

    # if we can, lighten the color for the subtree
    if alterColor:
        # make it lighter, but keep it between 0 and 1
        r = max(min(r0+0.2, 1), 0)
        g = max(min(g0+0.2, 1), 0)
        b = max(min(b0+0.2, 1), 0)
        self.color = (r, g, b)

    # draw right subtree
    self.drawBinaryTree(xpos=xpos+self.distance*scale*3.0/4.0,
                        ypos=ypos-self.distance*scale,
                        zpos=zpos+self.distance*scale/4.0,
                        scale=0.5*scale, numLevels=numLevels-1)

    # draw left subtree
    self.drawBinaryTree(xpos=xpos-self.distance*scale/4.0,
                        ypos=ypos-self.distance*scale,
                        zpos=zpos+self.distance*scale/4.0,
                        scale=0.5*scale, numLevels=numLevels-1)

    # restore the color
    if alterColor:
        self.color = (r0, g0, b0)
```