Assignment #2 (30 marks)

Due date: November 4th at 11:59 pm.

Problem 1 (22 marks): Playing Tetris with a robotic arm

You will upgrade the Tetris-FallingFruits game in assignment 1 into 3D, and control a robot arm to play this game. Check out the sample code for a robot arm in:

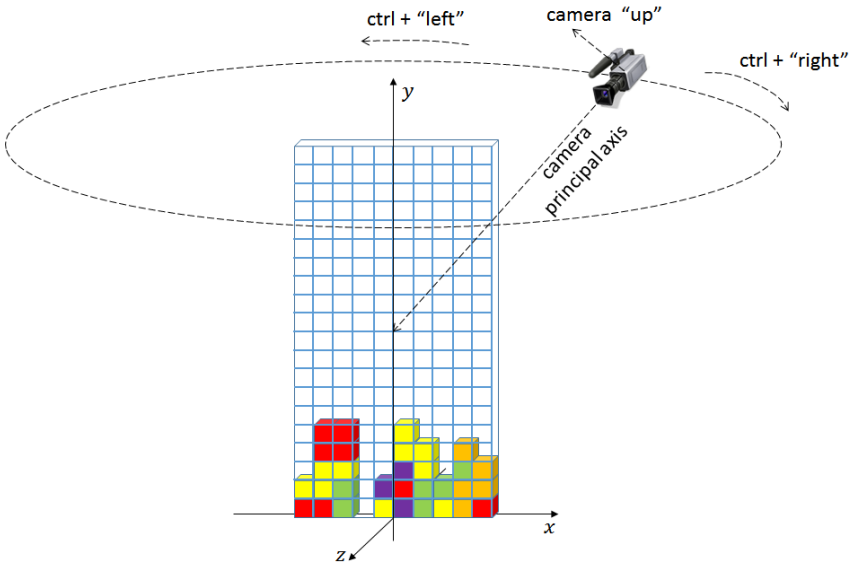
http://www.cs.unm.edu/~angel/BOOK/INTERACTIVE_COMPUTER_GRAPHICS/SIXTH_EDITION/CODE/CHAPTER08/
and once there, look into example1.cpp and associated shaders. You are allowed to use this code (or parts of it) in your assignment.

(a) [4 marks] Upgrading Assignment 1 to 3D

Upgrade your Tetris-FallingFruits game to 3D by turning each fruit from a 2D square to a 3D cuboid. The centers of all fruit cells are constrained in a 2D plane, e.g. $z = 0$. Draw the entire 3D grid--1 by 10 by 20--using faint lines. You can keep the same game logic at this stage.

(b) [4 marks] Viewpoint Changes

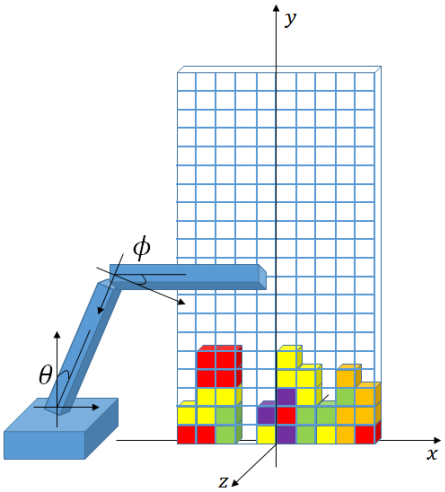
The "ctrl" key will be combined with the "left" and "right" key to control the camera's movement on a circular orbit that is parallel to the x-z plane with a center on the y axis. The camera always looks at the center of the Tetris window. Its "up" direction lies in the plane spanned by the y axis and the camera's principal axis – the line connecting the camera center and the "Lookat" point. Ensure that the camera always sees the entire board.



(c) [4 marks] Control of the Robot Arm

Put the robot arm in the 3D space of Tetris tiles. The central axes of both lower and upper arms will be limited in the same plane as fruit cuboids. In this way, the lower arm is controlled by a rotation angle θ, and the upper arm is controlled by an angle ϕ. Use the "a" (or "d") key to increase (or decrease) the angle θ, and the "w" (or "s") key to increase (or decrease) the angle ϕ. If you needed the characters "w", "a", "s", and "d" for block control

in assignment 1, then use "I", "j", "k", and "l" for the robot arm instead. Set the arm length and base height appropriately, such that it can reach **every** location of the Tetris window.



(d) [4 marks] Game Logic

New Tetris tiles only appear at the tip of the robot arm. Rotate the robot arm to put it at desired locations. At this stage, it is OK to ignore the collision between the new tile and existing ones. Use the "space" key to drop a Tetris tile from the robot arm. The tile's position should align with the grid, i.e. snapped to the nearest grid position. Once a tile is dropped, it can either float at the location it is dropped, or keep dropping downward until it hits some other tiles (like in Assignment 1).

The "up" key is still reserved to rotate a tile. The "left" and "right" keys are disabled. Press 'q' to quit and 'r' to restart. Eliminating full rows and three fruits with the same color is optional and will not be judged since it is evaluated in Assignment 1.

(e) [4 marks] Collision Detection

The new Tetris tile can only be dropped at places without collision with existing tiles. When the tile is attached to the robot arm, we allow collision, but will highlight collision cases by turning the tile to grey color (so that the player knows it cannot be dropped).

Also, if any cell of the tile (when its location is rounded to the nearest cube of the unit grid) is outside the playing area, turn the tile grey.

(f) [2 marks] Timer

The placement of each tile is limited to a certain amount of seconds. Display the remaining seconds (as a text message) for the current tile on the top of the window.

When the time is out, the tile on the arm will be automatically dropped at the current location. If the tile is greyed when the timer expires, then the game is ended. (But not your application…when a game ends it should be possible to play another game.}

Note that the above steps build on top of each other, in order. You need not submit individual programs to correspond to these steps. If you can implement all the required parts, a single, complete program is sufficient.

Problem 2 (3 marks): Back face culling.

Suppose we have a scene with a single, fully opaque, convex object, which is entirely inside the view volume. The convex object is specified by a mesh.

Under orthogonal projection, will back face culling alone be guaranteed to produce the correct set of visible polygons? Why or why not?

Under perspective projection, is it true that back face culling alone is guaranteed to produce the correct set of visible polygons? Why or why not?

Problem 3 (5 marks): BSP vs. depth-sort.

Show that the back-to-front display order determined by traversing a BSP tree is not necessarily the same as the back-to-front order determined by depth-sort, even when no polygons are split. To receive full marks on this problem, the number of polygons you use for your example must be the smallest possible and you also need to prove that the number of polygons you used is the smallest possible. **Hint: mostly you need an example. The rest is easy.**

Prepare a PDF file ***problems.PDF*** with your answers to problems 2 and 3.

Submission: All source code, a ***Makefile*** to make the executable called ***FruitTetris3D*** (the make command should be simply "make"), a ***README*** file that documents any steps not completed, additional features, and any extra instructions for your TA, and your ***problems.PDF*** file.