# Computer Science 474 – Computer Graphics

**Spring 2014**

**PEX 5 - 120 Points**

**Due: Lesson 34, Wednesday, 23 April, 2014 @ 2300**

# Help Policy

|  |
| --- |
| **AUTHORIZED RESOURCES:**   Any, except another cadet’s program.  **NOTES:**   * Never copy another person’s work and submit it as your own. * Do not jointly create a program. * You must document all help received from sources other than your instructor or instructor-provided course materials (including your textbook). * DFCS will recommend a course grade of F for any cadet who egregiously violates this Help Policy or contributes to a violation by others. |

# Documentation Policy

|  |
| --- |
| * You must document all help received from any source.  Unless quoting directly or paraphrasing, you do not need to document your course text, lectures, or any other course materials provided by your instructor. * Each documentation statement must be specific enough that it explicitly describes **what** assistance was provided, **how** it was used in completing the assignment, and **who** provided the assistance. * If no help was received on this assignment, the documentation statement must state “NONE.” * If you checked answers with anyone, you must document with whom on which problems.  You must document whether or not you made any changes, and if you did make changes you must document the problems you changed and the reasons why. * Vague documentation statements must be corrected before the assignment will be graded, and will result in a 5% deduction on the assignment. |

# Learning Objectives:

* To learn how to implement lighting in a 3D rendered scene.
* To learn how to implement texture mapping on 3D objects.
* To add game logic to your 3D Tetris game.

# Project Description

This is the fifth programming exercise (PEX) in a series of five PEX’s that will create a 3D Tetris game.

For this PEX, please enhance your PEX 4 solution to include: 1) at least one 3D object that is texture mapped, 2 ) a point light source that illuminates the models, 3) the display of the “widget on deck” in a 2nd canvas window, and 4) additional game logic to finish the Tetris game.

# Detailed Requirements

**Task 1: Texture Mapping**

* Include at least one object in your Tetris scene that is texture mapped.
* You will need to add texture coordinates to at least one of your 3D models using Blender.
* You will need to use a new version of the Model3DCreator.js code, called Model3DCreatorV3.js. This will require you to change your vertex and fragment shaders appropriately. Get the new software and new shaders from the link on the PEX5 assignment page.

**Task 2: Lighting**

* Include a *point light source* in your 3D Tetris game.
* Place the light in a location that nicely illuminates the 3D models in your game.
* Modify your WebGL *fragment shader* program to use the point light source to calculate the color of individual pixels. (Note that the data needed by the *fragment shader* must be set up in the *vertex shader*.)
* Optional: It is common to move lights around in a scene to add “excitement” to the scene. Think about a TV or Broadway show where spot lights are constantly panning over the stage and crowd. You can make your Tetris game more visually interesting by animating the location of the point light source, but this is totally optional.

**Task 3: Display the “next” widget**

* Display which widget will fall next after the currently falling widget stops. Display the widget in the 2nd canvas window of your web page.
* For the rendering of this canvas window, you will need a *projection matrix* and a *camera matrix*. (You don’t need a *model matrix*, but you can use one if you want to.) Set the camera matrix so that the user gets a good 3D view of the widget. Set the projection matrix to either a perspective or an orthogonal projection – it doesn’t really matter for this view.
* Optional: It would be a common feature in a game like this to have the widget spinning inside the window. Spinning the widget guarantees that the user can see the true 3D nature of the object. In addition, spinning the widget adds visual activity to the screen which potentially makes the game more enticing. You can spin the widget using a model matrix to rotate the object, or you can get the same visual effect by moving the camera around a stationary widget. Again, animating the widget so that it is spinning is optional.

**Task 4: Tetris Game Logic**

* Implement at least 2 of the following game features. Note that some features are built on top of other features, so please make sure you only implement features that have the required functionality included. For example, don’t implement #2 unless you have implemented #1 first.

|  |  |  |
| --- | --- | --- |
| **Feature** | **Description** | **Requires** |
| 1. Score | Calculate a score and continuously update it on the web page. (See <http://tetris.wikia.com/wiki/Scoring> for a possible scoring system.) You may use any scoring system you want to. | -- |
| 2. High Score | Remember the highest score obtained and the player’s name that made the score. Display this information on the web page. The “high score’ is considered the highest score obtained during the current play session. (To do this over multiple users and multiple play sessions requires that the server keep the state of the game. The web page has to send the high score to the server so it can permanently store it. This is beyond the scope of this class.) | #1 |
| 3. Level | Modify the difficulty of the game over time and display the current “level” on the web page. Modify the difficulty by making the widgets fall faster. There are many possibilities for when the game becomes harder, including 1) a certain number of widgets have fallen, 2) a certain score has been obtained, 3) a certain amount of time has expired, or 4) a certain number of user translations and rotations have occurred. You can use any of these criteria (or any other criteria you design.) | -- |
| 4. Start Over | Restart the game with an empty container. The score, high score, and level should all be reset appropriately (if they exist), as well as the state of any control buttons. | -- |
| 5. Game Over | The current game stops when it is not possible to add a widget to the container. The state of the game should be updated to indicate that the game is over, such as disabling the “pause/resume” button because it should not be active if the game is not running. | -- |
| 6. Other? | If you have a game feature that you would like to add and it is not in the above list, please suggest it to your instructor. *If your instructor approves* of the proposed feature, you can implement the feature for this PEX. | -- |

# Turn-in

Electronic turn-in: (at <http://dfcs-moodle> on the course web site). Please submit all files needed to execute your Tetris 3D game web page. If the number of your files exceeds the upload file limit, please submit a single zip file that contains all of your files.

**Name:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**CS474 – PEX 5 - Cut Sheet**

**120 Points**

|  |  |  |  |
| --- | --- | --- | --- |
| **Grading Criteria** | **%** | **Pts** | **Comments** |
| **Task 1: Texture Mapping** | **25%** | **30** |  |
| Texture coordinates are defined in at least one model definition. |  | 10 |  |
| Appropriate code is included in the shaders to render texture maps. |  | 10 |  |
| One 3D object is visibly texture mapped on the screen. |  | 10 |  |
| **Task 2: Lighting** | **30%** | **36** |  |
| One point light source is implemented in the scene. (The light source is not visible, but the effects of the light are visible.) |  | 16 |  |
| The lighting calculations are done on a pixel by pixel basis in the fragment shader (with appropriate setup in the vertex shader and JavaScript program). |  | 20 |  |
| **Task 3: Display Next Widget** | **20%** | **24** |  |
| The “next widget” is displayed in a 2nd canvas window. |  | 14 |  |
| A reasonable projection matrix is used. |  | 5 |  |
| A reasonable camera (view) matrix is used. |  | 5 |  |
| **Task 4: Tetris Game logic** | **20%** | **24** |  |
| Feature A |  | 12 |  |
| Feature B |  | 12 |  |
| **Software Design and  Good Coding Style** | **5%** | **6** |  |
| **Total** |  | **120** |  |
|  |  |  |  |
| Poor or missing documentation (-5%) |  | -6 |  |
| Late Penalty Cap  (25% per 24 hour period) (90, 60, 30, 0) |  |  |  |
| **Final Grade** |  |  |  |