6.111 Final Project Proposal: 3D Tetris

Charity Midenyo, Jenessa Rodriguez, Bradley Seymour

MIT EECS

Abstract

A working Tetris game (“random” next block, falls down and stacks, if a row is full, it is cleared, etc). The playing field will have depth and the ability to rotate the playing field in all directions. We will use a controller to play the game, so that will have to be mapped. And no Tetris game is complete without the short song being played. Stretch goals are to create various challenge modes and a working title screen.

6.111 Final Project Proposal: 3D Tetris

35 years ago, the world was introduced to the Tetris[1]. Since then, Tetris has grown exponentially, with all various conceivable versions having been created. One of the more rare versions of Tetris would be the 3D version. 3D Tetris was first introduced to the world in 1996 on Nintendo's Virtual Boy console[2]. The failure of Virtual Boy as a console notwithstanding, the game itself attempted to revolutionize the way Tetris was played. The next version of a 3D Tetris would be presented to the world as Tetrisphere. Released in 1997[3] for the Nintendo 64, Tetrisphere proved the concept of a 3D version of Tetris to be viable. Assuming an appropriate representation of the playing field and control system implemented by the developers, a new standard for how 3D Tetris should be played was born. Our team intends to create a version of 3D Tetris, utilizing these games as inspiration for our personal take on how 3D Tetris should look, play, and most importantly, feel.

Implementation

Our game will be built utilizing a Xilinix Nexys 4 DDR FPGA board to create the game logic, graphics system, and I/O control. After research into available gamepads, our team settled on use of Sony's Dualshock 4 (DS4) gamepad. The decision to utilize a DS4 was born from the convenience of availability coupled with finding reports on the organization of output data and report generation rates utilized via USB mode. Our team identified three major aspects requiring creation to bring 3D Tetris to life, the game logic, a 3D graphics engine, and I/O logic modules. Each major area of design will further be broken into smaller modules as follows.

---

[1] (Wikipedia: Tetris, n.d.)
[2] (Wikipedia: 3D Tetris, n.d.)
[3] (Wikipedia: Tetrisphere, n.d.)

Game Logic

Three FSM modules will lay at the heart of the game: block movement, line removal and scoring (LRS), next block generation. The block movement FSM will decide based on gamepad input how to position a tetromino within the playing field. As with all Tetris games, holding down on the D-Pad will increase the falling velocity. Data from the block movement FSM will be sent to the LRS FSM to determine when a line needs to be removed from play. The LRS FSM will also tally score for placed tetrominos and cleared lines. When the LRS FSM returns to idle, the next block FSM will activate. Taking a pseudo-random number from the generator, the next piece can be chosen for the player. Upon completion of the update state, the game logic control module will pass a 3D array containing all relevant RGB data for the playing field to the 2D/3D graphics engine.

Module List.

1. Game Piece Movement FSM
   a. Inputs: Button Presses, 8bit
   b. Outputs: Movement Direction State
      10x20 Array, X, Y format, 12bit RGB
   c. Function:
      1. Move the piece down (whether naturally or controlled)
      2. When to stop moving down
      3. Move left
      4. Move right
      5. Rotate
2. Line Removal and Scoring FSM
   a. Inputs: Game Piece Movement FSM State
   b. Outputs: Score, 16bit
      10x20 Array, X, Y format, 12bit RGB elements
      LRS FSM State
   c. Function:
      1. Remove one line
      2. Remove two lines
      3. Remove three lines
      4. Removed four lines

5. Move the lines above down
6. Update the score accordingly

3. Next Block FSM
   a. Inputs: LRS FSM State
      RNG Value, 16bit
   b. Outputs: 10x20 Array, X, Y format, 12bit RGB elements
   c. Function:
      1. Generate the next piece
      2. When to send the next piece

3D Graphics Engine

Representing a 3D object on a 2D screen is no easy feat. The human brain must be tricked into seeing 3D space through use of appropriate scaling and shading of 2D objects. The triangle is chosen as the simplest 2D object that can be drawn and accurately scaled to a given field of view to maintain a 3D perspective. These calculations are to be performed utilizing 16 bit numbers to represent each x, y, and z point in space. The 16 bit number will be broken into a 12bit integer with a 4 bit mantissa to allow for pseudo floating point operations to be performed. These values are subject to change to allow for greater precision when drawing the game field should it be required. To support the calculations, a matrix math module will be created for ease of coding in Verilog. The goal is to create sufficient modules and arrays such that the Verilog code can be treated in a manner similar to a higher level coding language to create the 3D field. As a stretch goal, texture maps will be utilized to replace the RGB color values for each 3D object drawn.

Module List.

1. Game Board to Mesh
   a. Input: Array, 10 x 20 elements, 12bits per element representing RGB values
   b. Output: Vector of 12bit X,Y values
      Vector 12 bit RGB values for each cube at (X, Y) position
   c. Function: Steps through 10x20 Array, creating 2 Byte X and Y values from each
2. Triangle from Mesh
   a. Input: Vector of (X,Y) values
   b. Output: 10x20 Array of (X,Y) vectors representing each field

   c. Function: Takes the playing field
3. Triangle transformation
   a. Input Array of vectors for each triangle vertex
   b. Output: Array of vectors for each triangle vertex
   c. Function: Each vertex arrives as a 3D point that needs to be projected to a 2D field. This module will scale and modify each field to create the projection of a 3D shape to 2D space.
4. Matrix Math
   a. Input : 4x4 Matrix, 1x4 Matrix
   b. Output: 1x4 Matrix
   c. Function: Performs necessary calculations on a given set of coordinates to scale appropriately for "3D" effect. Returns actual location of 3D point.
5. Draw shape:
   a. Input: 3 1x4 matrix representing a single triangle
   b. Output RGB color value
   c. Function: Transform a triangle represented by 3 1x4 matrices into RGB color space.

Input/Output

On their own, input and output from systems is often overlooked and forgotten about. Our game will require several independent IO modules to properly function. The design calls for the use of a Sony Dualshock 4 (DS4) as the input gamepad. The intention is to use the D-Pad to control block movements around a "2D" plane, similar to the way the original Tetris is played. Additionally, two of the 4 buttons will be utilized to rotate a block within 2D space. Finally, the analog sticks on the DS4 will be used to rotate and zoom the game board in 3D space. The UI will be drawn in 2D space, and for early testing purposes, a 2D game board will be created to enable testing of game logic. If time allows, an audio system will be implemented to incorporate the traditional songs of Tetris as well as block collision sounds and line clearing sounds. These files will be stored on flash RAM for access by the game as required. A COE file of ASCII values will be stored in BRAM for access and use by the scoring subsystem. Separate COE files will be utilized to display the next piece the user is will receive.

Module List.

1. Controller Inputs
    a. Input: DS4 input - once every 4ms
    b. Parses the data dump of the controller to get the information necessary to the game and rotation of the screen
2. Game controls
    a. Inputs: down left, right, rotate movement of block
    b. Output: byte with the first 4 bits representing the movements above (if rotate button is pressed, the 4th bit would be a 1)
3. Rotational controls
    a. Input: 4 byte values representing up, down, left, right , rotation of the tetris board
    b. Output: 200 wide array of 12 bit representing x,y,z of each tetris block in the game
4. 2D output display
    a. Inputs :
        1. 200 wide array of the location of each block
        2. 200 wide array of the color of each block - can be black
        3. Player score
        4. Game level
        5. Next piece
    b. Output - renders the tetris game board in xvga screen, with all of the blocks and score, level and next piece
    c. Part of this module output will be overwritten by the 3d module, but it will be useful to have a working 2d rendering of the game for testing

Deliverables

The following events will serve as proof for each portion of the game working:

Input-Output Modules

- Being able to see and process controller data as lights flashing on the FPGA
- Calculating Z location for rotation around 1 Axis - show on the FPGA digits
- Calculating Z location for rotation around All Axis - 3D display
- Draw Tetris Screen- UI and 2D play screen
- Update screen from Game module- UI and 2D play screen
- Play game with controller in 2D

Game Logic.

- Show automatic falling block.
- Falling blocks stop when striking bottom of screen or block.
- Move a block down, left, right, and rotate.

- Clear a line when full.
- Clear multiple lines when full.
- Update game score when lines cleared.
- Update game score when single piece completes movement.
- Be able to send data to 2D playing field
- Choose, create, and send next block to 2D playing field.
- Create the next block after scoring and line removal.

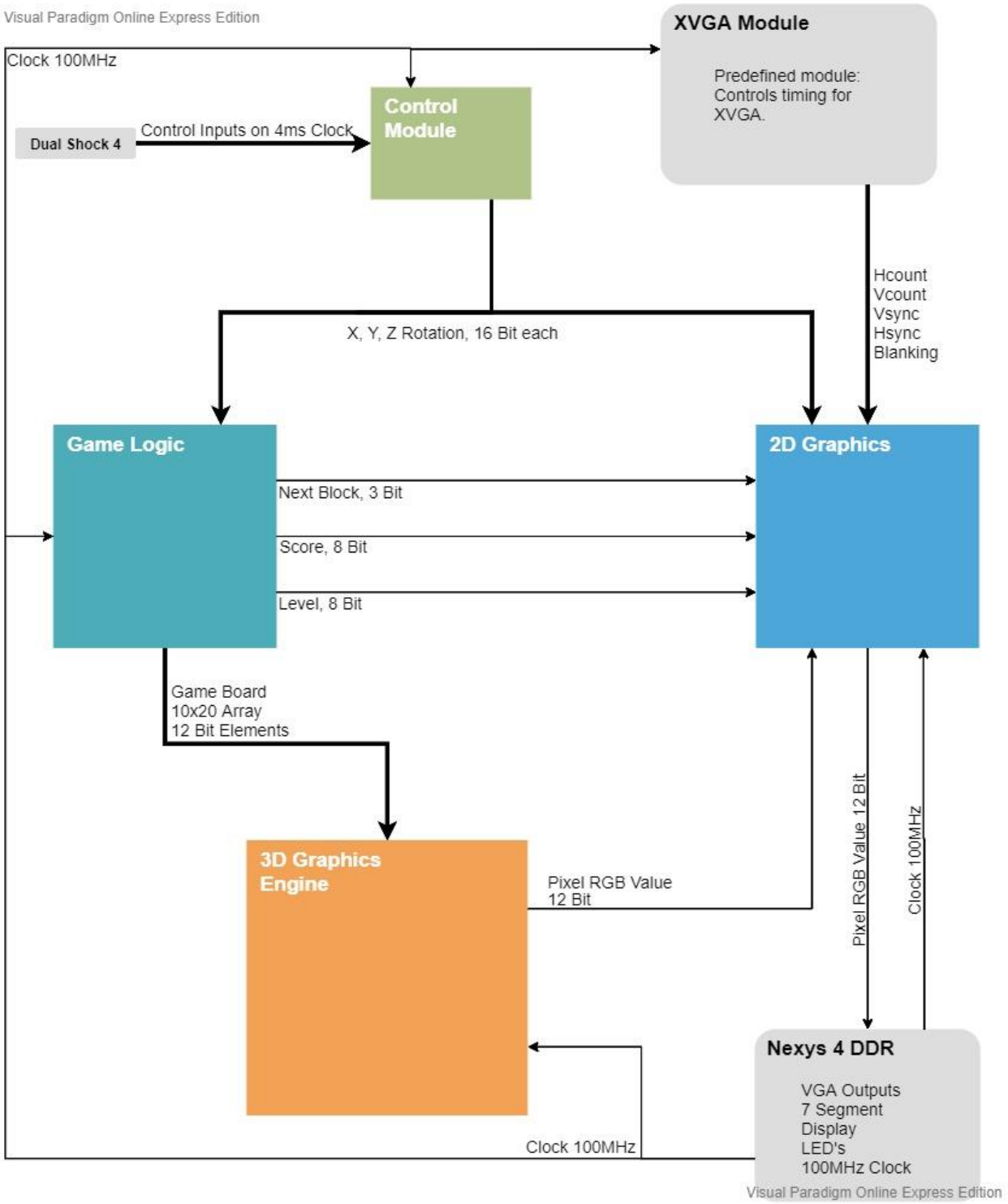
Sound Engine.

- Play Tetris sound

3D Graphics Engine.

- Matrix math modules, known inputs display correct outputs in simulation
- Draw a single block in 3D space
- Rotate a block in 3D space
- Build a 3D playing field of tetronimos
- Rotate 3D field about single axis
- Zoom field along Z-axis
- Draw a 3D field based on given input data
- Rotate field in 3D space and zoom

Block Diagram



References

Wikipedia: 3D Tetris. (n.d.). Retrieved 10 27, 2019, from https://en.wikipedia.org/wiki/3D_Tetris

Wikipedia: Nintendo Entertainment System. (n.d.). Retrieved 10 27, 2019, from https://en.wikipedia.org/wiki/Nintendo_Entertainment_System

Wikipedia: Tetrisphere. (n.d.). Retrieved 10 27, 2019, from https://en.wikipedia.org/wiki/Tetrisphere

Footnotes

[1][Add footnotes, if any, on their own page following references. For APA formatting requirements, it's easy to just type your own footnote references and notes. To format a footnote reference, select the number and then, on the Home tab, in the Styles gallery, click Footnote Reference. The body of a footnote, such as this example, uses the Normal text style. *(Note: If you delete this sample footnote, don't forget to delete its in-text reference as well. That's at the end of the sample Heading 2 paragraph on the first page of body content in this template.)*]

Tables

Table 1

[Table Title]

Column Head	Column Head	Column Head	Column Head	Column Head
Row Head	123	123	123	123
Row Head	456	456	456	456
Row Head	789	789	789	789
Row Head	123	123	123	123
Row Head	456	456	456	456
Row Head	789	789	789	789

Note: [Place all tables for your paper in a tables section, following references (and, if applicable, footnotes). Start a new page for each table, include a table number and table title for each, as shown on this page. All explanatory text appears in a table note that follows the table, such as this one. Use the Table/Figure style, available on the Home tab, in the Styles gallery, to get the spacing between table and note. Tables in APA format can use single or 1.5 line spacing. Include a heading for every row and column, even if the content seems obvious. A default table style has been setup for this template that fits APA guidelines. To insert a table, on the Insert tab, click Table.]

Figures title:

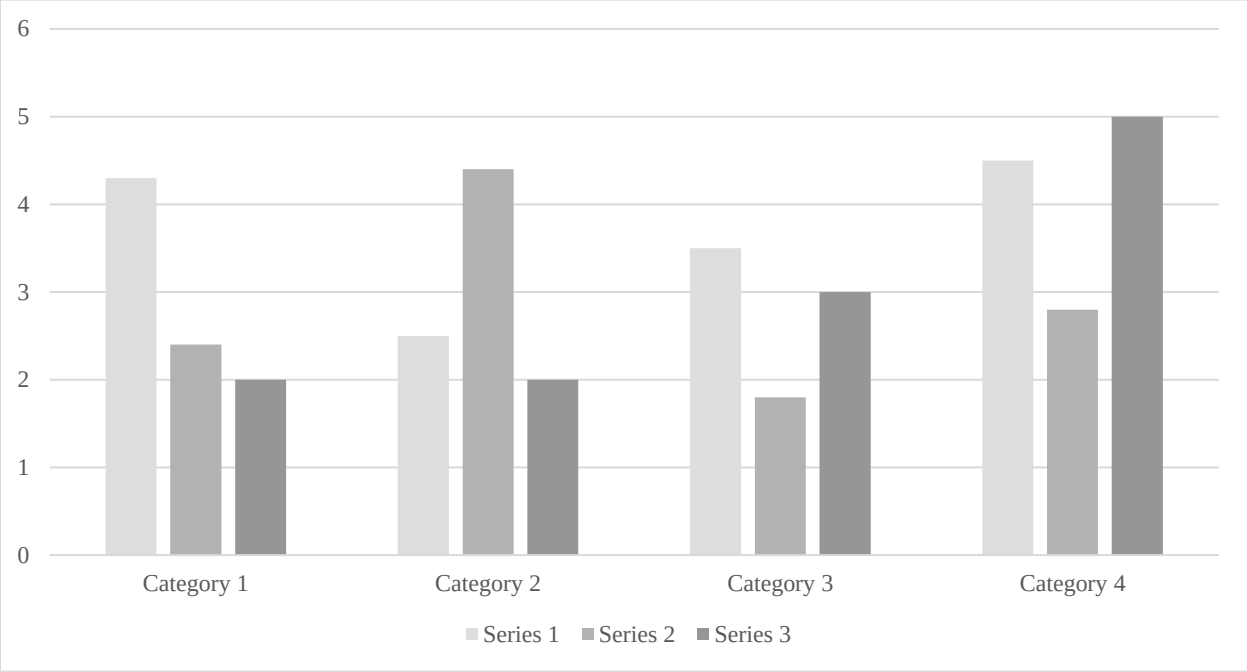


Figure 1. [Include all figures in their own section, following references (and footnotes and tables, if applicable). Include a numbered caption for each figure. Use the Table/Figure style for easy spacing between figure and caption.]

For more information about all elements of APA formatting, please consult the *APA Style Manual, 6th Edition*.