3D Tetris

COS 426 - Computer Graphics

Anthony Gartner, Edward Gartner, and Daniel Goodman

Abstract

For this 3D Tetris project, our goal is to craft a three-dimensional version of the classic game of Tetris. While in traditional Tetris, 2D pieces made of four squares each are stacked up with the goal of filling up complete horizontal lines, 3D Tetris has the goal of stacking up 3D pieces (each made of four cubes) with the goal of filling up horizontal planes. To achieve this goal, three.js was used to create a web app that allows this game to be played. As of the submission of this write-up, the project is nearing a complete minimum viable product, which implements random falling pieces, collision detection, piece and camera control, and a plane-clearing and scoring system.

Introduction

Goal

With this project, we tried to create a faithful representation of what classic Tetris would look like in three dimensions. We also placed a focus on intuitive, easy-to-grasp gameplay, so the player can jump right in without having to learn complicated controls. From the start of the project, it was important to us that the movement and rotation of the pieces felt natural.

This project would benefit fans of the original Tetris who want to try out a new challenge and potentially improve their 3D spatial reasoning skills while having fun. This would also benefit those who enjoy relatively fast-paced, skill-based puzzle games, as the game essentially acts as a skill that can be honed and improved over time. For players who enjoy keeping track of high scores and seeing their own improvement over time, this game will be a great match.

Previous Work

In the previous semester, a group also attempted a 3D Tetris project. In this particular instance, the final result was much more reminiscent of classic 2D Tetris, but with 3D assets. In 1996, there was a release of 3D Tetris for Nintendo's short-lived Virtual Boy console, which featured stereoscopic 3D graphics and similar gameplay to that of our project. However, this version of Tetris (and the Virtual Boy as a whole) was not well-received at the time due to being "eye-straining" and generally feeling unfair ("3D Tetris", "Virtual Boy").

While the first of these two approaches was successful as a Tetris game, it does not implement the same features we hope to implement. The second example seemed to be a failure less due to the idea, but more due to the painful viewing experience and the difficult controls ("3D Tetris"). To avoid issues like these, we have focused on making an intuitive camera and control system that will leave the player feeling stimulated as opposed to frustrated.

Approach

To approach this goal, we started by designing and modeling the different types of pieces. Taking a page from classic Tetris, we wanted to include pieces that encompass every possible permutation of four cubes. This resulted in a few surprises: for one, in three dimensions, the classic L and J pieces are the same due to the new 3D rotation (as are the S and Z pieces), so these have each been merged into one. Additionally, there are three new types of pieces (dubbed W, A, and V), which each encompass all three dimensions. This set of pieces seemed like it would work well since we end up with a manageable total of eight different types of pieces, so no piece should ever be too rare; this is comparable to classic Tetris, with its seven pieces. Some of the piece models are shown below.

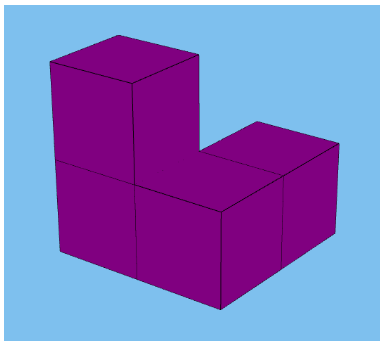
Figure 1: New “A” piece

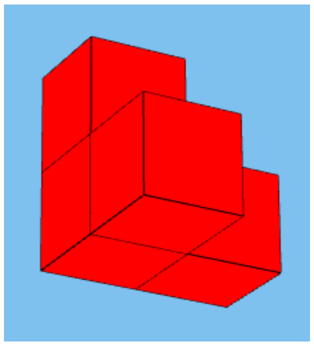
Figure 2: New “W” piece

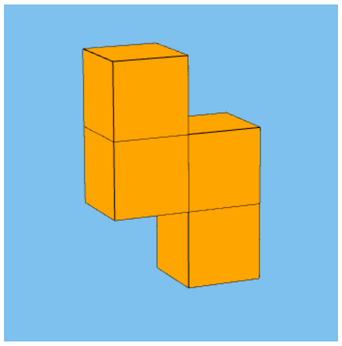
Figure 3: Classic “S/Z” piece

After designing the pieces, we designed the camera, which is purposefully limited to always keep the playfield in the middle of the screen, but simultaneously allows for 360-degree rotation around it using the mouse. This allows for one hand to be used to control the rotation and movement of the pieces, while the other hand controls the camera with the mouse. This seems logical, as the mouse provides a very intuitive and free-feeling analog camera mechanism, whereas the keyboard is ideal for the more digital movement of the pieces. This will work best after the player gets some practice with the controls. We anticipate that at first, it may be difficult to remember which keys are mapped to which rotation axes, but it should become second-nature after some practice.

Finally, the rules of the gameplay itself had to be added, such as the size of the playfield, the points system, and the decision to require the player to fill up entire planes instead of lines. The playfield is currently set to a 4x4x15 grid, although these exact dimensions may change for the final version. We believe that these dimensions will be good for the sake of having a relatively small set of choices for each piece, so the player is not overwhelmed. With a larger base (6x6x15 or greater), we feel that it would be very time-consuming and tedious to clear a plane, which would make the game less captivating in general. With our current choice, we believe the gameplay will feel cleaner and punchier. As far as the decision to require the player to clear planes instead of lines, this simply seemed to be the most logical step into three dimensions. We want the player to be thinking in 3D while playing this game, so having them clear planes seemed like the best way to achieve this goal. However, while not the goal of this project, it could be interesting to see a version of 3D Tetris where 1xN lines have to be cleared instead of NxN planes.

Methodology

The first part of executing our approach was creating the Tetromino objects, the playing pieces, for the game. We had the choice between modeling each of the eight types of pieces, along with all possible iterations of those models after cubes are cleared, or we could use cubes gathered into groups. Modeling each iteration would have only brought disadvantages, as we would have had to do much more work both in modelling, and in replacing each piece with other iterations as the game progressed, so we chose the vastly more advantageous option of making the tetrominos with groups of four cubes, as this allowed us to rotate and translate the shapes all together, while still providing the functionality to eventually separate the cubes from each other and delete cubes. We used this method to implement eight separate piece models, which have translation controlled by the W, A, S, and D keys, and three dimensions of rotation controlled by the Q and E, R and F, and Z and C keys, as well as collision detection. We are currently working through an issue where the rotation is not properly updating the boundaries of the pieces, yet still updates the visual position of the pieces.

Another significant piece of our implementation is the camera. There were numerous ways we could have implemented the camera, such as remaining with the default orbital camera, allowing the player to switch between several fixed camera positions, having a fixed camera, or having a custom, modified orbital camera. The orbital camera provided the best range possibilities for vision, but required the player to remember to click and drag to move the camera, which we decided would be clumsy while trying to play the game. The fixed camera variations could have had simpler camera controls, but offered less range of vision, and we wanted to be sure to implement this in as robust 3D as possible. We chose the third option, the custom orbit camera, because it allowed a wide range of vision with extremely simple controls. This custom orbit camera works by locking the mouse cursor once the game begins, and using the mouse's movement alone to move the camera, which allows the player to focus entirely on moving pieces, while naturally moving the camera.

The next major portion was the movement of the pieces. This was briefly discussed above, and our only choices for implementation were either making the controls do only fixed rotation and translation of the pieces, or making the keys manipulate the pieces based on the camera orientation. We decided on implementing it based on the orientation of the camera as it would allow for more easily understandable gameplay and more natural piece movement at the expense of slightly more computations. We are currently working through an issue where the pieces will move in the inverse direction at high-overhead angles of the camera.

Another part was the random generation of tetrominoes, which we accomplished by assigning each type of piece a letter to represent the piece, then chose a random letter from a string of all the letters, then generated the piece in a specified initial position, and added it to an array of all the current pieces. The piece would then be translated down one position every certain number of frames in order to give the player time to react and move the piece, and to recreate the classic movement of tetris pieces.

We also created a menu to navigate between relevant pieces of information about our project, including information about the COS 426 Final Project in general, the credits for the project, the instructions of how to play the game, and a button to start the game itself. We found a menu screen such as this was necessary both to introduce controls to new players, and also to give players a moment to ready themselves, as Tetris relies on readiness and planning, so we didn't want to surprise players that weren't ready. The menu was implemented with a series of THREE.js Scenes that had css elements to display the controls, credits, project info, etc.

Finally, we implemented a system for keeping track of the position of each cube on the board, alongside a system for detecting and clearing completed planes, and awarding points. We have two current possible implementations for keeping track of the positions, both of which use a three dimensional array to represent the game board. The current implementation resets the entire array and fills it again based on the position of each piece, every time a piece is moved and manipulated, summing the total number of cubes on each plane once the positions are updated. This approach is simple and worked best with our current implementation of other features, but is drastically inefficient. We are planning to change the system to instead use the previous position of each tetromino to clear out only a select number of values in the array instead of the entire array, which would allow us as well to better keep a running count of cubes on each level that only updates when a piece comes to rest. As soon as we edit some other code, we will change to the more efficient and robust implementation, but we settled with this implementation for the time being in order to avoid creating issues with older code before Dean's Date. Additionally, the plane-clearing and score mechanics haven't yet been actually used in our code, as due to a bug with collision detection, planes cannot yet be filled anyway. The scoring mechanic works by, when a piece reaches its resting place, the update code clears how many planes can be cleared, and depending on the number of cleared rows, points are awarded in a similar fashion to the original Tetris. We also have implemented the start of a level-progression system based on the amount of rows that have been cleared so far, but we haven't yet fully implemented it, as rows cannot yet be cleared, and we haven't yet completely decided the specifics that a new level of difficulty would entail.

Results

Fortunately, as we were implementing a game, it was relatively easy to measure success. We needed to match the game constraints that the original Tetris had set, but bring it to a 3-dimensional landscape. As such, our project was highly constrained to a very specific ruleset. If the project deviated from this ruleset, we had a choice to either implement the deviation as a feature or correct it. For the most part, we opted to correct the difference, although we have not totally solved all of our bugs yet. For example, collision detection in a normal sense is unnecessary because there are a finite, small number of potential board positions for tetriminos to inhabit, and piece movements are quantized rather than smooth. Thus, we only needed to verify that a piece was not about to move into another piece's space.

Along these lines, most of the experimentation we undertook was using methodologies different from those used in most of the assignments. Whereas the assignments featured smooth movement and increasingly detailed meshes, our project instead strives to give a simplistic yet fun experience with jumping movement and simple shapes. Thus, a challenge moving forward will be providing an aesthetic and efficient experience for players. Not much experimentation is necessary with gameplay mechanics given that we are just bringing a popular game to a 3D space, but given that we have a gaming titan to live up to, we need to be flexible and experiment with aesthetic stretch goals now that our MVP is basically complete.

Discussion

Overall, the work we've done thus far is promising. Moving forward, we are a few bugfixes away from a fully functional MVP. Specifically our next steps will be to fix the issues with collision detection and rotation, and with the camera and translation, then we will change code to better implement the methods of keeping track of the location of each cube and clearing planes of cubes. Past those bug fixes, we can then go on to tackle finalizing and displaying the score and level systems, as well as improving the aesthetics of the game.

Throughout the project so far we've learned a number of things. First, we learned how to effectively use groups to efficiently manipulate groups of objects at once, and we learned how to continuously and automatically generate 3D objects. We also learned how to better use event handlers and the Pointer Lock API to create more useful and natural controls. Finally, we learned how to use scene transitions as an effective menu.

Conclusion

Our goals were first to make sure what we created fully utilized the 3D setting, which we believe, through our reimagining of a 2D game in 3D, complete with new pieces and mechanics, along with our effective camera system, we have successfully attained one of our goals, though we do not yet have our MVP. Our next steps will be to fix the remaining bugs between our current progress and MVP, then to add a score and level system, and increase the aesthetic appeal of our game.

Contributions

The work for this project was split up such that Daniel was responsible for the Tetromino piece design and basic gameplay elements (such as incremental piece drops and much of collision detection), Edward worked on the piece controls and start-up menu screen, and Anthony worked on the camera controls and the visual elements of the base and gameboard, as well as the score system and plane clearing.

Works Cited

"3D Tetris." *Wikipedia: The Free Encyclopedia*. Wikimedia Foundation, Inc., 30 Apr. 2021, https://en.wikipedia.org/wiki/3D_Tetris. Accessed 10 May 2021.

"Virtual Boy." *Wikipedia: The Free Encyclopedia*. Wikimedia Foundation, Inc., 5 May 2021, https://en.wikipedia.org/wiki/Virtual_Boy. Accessed 10 May 2021.

Additionally, the following is a list of some of the web sources that were consulted while programming so far:

- Many pages of the three.js documentation
- https://github.com/MichaelF49/Pacman3D/blob/master/src/global/index.js
- https://github.com/MichaelF49/Pacman3D/blob/master/src/global/globals.js
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes
- https://developer.mozilla.org/en-US/docs/web/javascript/reference/statements/export
- https://github.com/mrdoob/three.js/blob/master/examples/jsm/controls/OrbitControls.js
- https://github.com/mrdoob/three.js/blob/master/examples/jsm/controls/PointerLockControls.js
- https://tetris.fandom.com/wiki/Scoring