3D Block Stacker Game Report

Xiaohui Chen
Department of Computer Science
The University of Texas at Austin
Email: xhchen0328@utexas.edu

Yichi Zhang
Department of Electrical and Computer Engineering
The University of Texas at Austin
Email: yichizhangmax@utexas.edu

***Abstract*—In this project, we implement a 3D block stacker game. We utilize the concept of hierarchical model to create a crane. We also use collision detection of OBBs to prevent inter-penetrations. With the help of ODE simulator, rigid body simulation in our game is possible. In face, our game is close to a realistic block stacking.**

1. Introduction

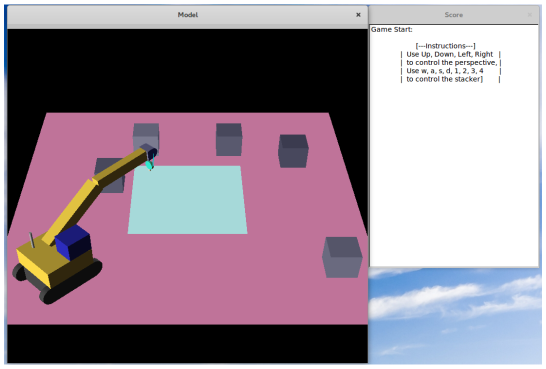
In this project, we built a game called block stacker. This serves as an extension from *Assignment 4*, however, with a lot of new feature added. In this project, we designed a new user interface, rebuilt and enlarged the hierarchical model, added the keyboard controlling function, implemented the texture mapping and made rigid body simulation of block stacking possible.

In order to win this game, the player should try to use the crane to pick up each stone block and pile up in the right region in the middle of the ground. The final score would be the height of the stack. The player with the highest score wins the game.

Section 2 of this report describes the user interface we modified. Section 3 provided information of our crane, which is created using the concept of hierarchical model. Section 4 provides information on how to use keyboard to play this game. Section 5 describes the outputs players could see. Section 6 describes the texture mapping techniques used in this game. Section 7, 8, 9 describe the two different spaces are used in this game and the corresponding collision detection schemes. Section 10 provides an insight to rigid body simulation. Finally, section 11 and 12 describes the possible results and future improvements of this game.

2. User Interface Modification

In our project, we want to reduce the remnant components irrelevant to our game. Therefore, we implemented keyboard controlling to replaced the controlling window. Thus, our game would be more laconic and easier to get started. Also, we implemented a new window as a billboard to show the instructions, current status of the game and obtained score. Once the status is updated, a new message would be prompted for recording purposes. The initial view is shown in Figure 1.



```
Game Start:

        [---Instructions---]
      |  Use Up, Down, Left, Right   |
      |  to control the perspective, |
      |  Use w, a, s, d, 1, 2, 3, 4  |
      |  to control the stacker]     |
```

Figure 1. Initial view of user interface

3. Hierarchical Model

To start with, we build a new hierarchical model for the crane. The hierarchical model is based on the robotarm in animator project. The main modification is the the claw part. We replaced it with a hook-block. We also created the stones. As extended from the animator project, we keep the basic shape of the crane but implemented more hierarchical layers and created more objects. In order to make the crane more realistic, we further built the wire ropes and the hook-block with a magnet at the end of the hook-block which can pick up stone blocks. In addition, the wire rope and the hook-block should be always drooping under no force influence but gravity. The details are shown in Figure 2.

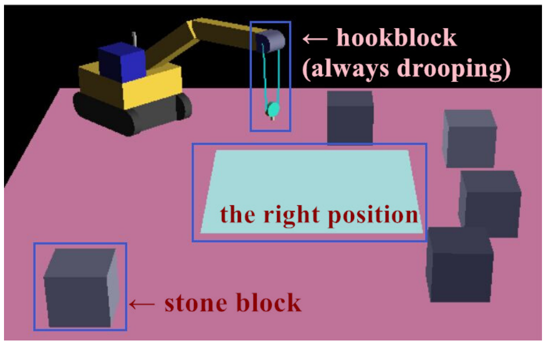


Figure 2. Detailed demonstration of each part

4. Keyboard control

As mentioned before, we implemented keyboard controlling to set several hot keys. *FL::event_key()* here is used to detect which hot key is pressed by the player. In our project, 17 hot keys are implemented to control both perspective and objects. Among them, 13 keys are used more frequently. Here *ESC* is used for resetting the game. *Up*, *Down*, *Left* and *Right* are responsible for controlling the perspective. *A* and *D* are used to control the angle of the arm. This means pressing those keys could make the arm to move left or right. In addition, *W* and *S* can move the hook-block up or down. *1* and *2* control the angle of the arm in order to move it to a higher or lower position. *3* and *4* control the length of the upper arm. Hot keys directions are shown in Figure 3, Figure 4, and Figure 5.

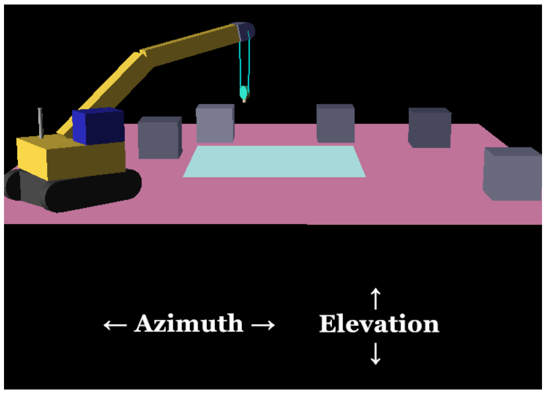


Figure 3. Controlling perspective

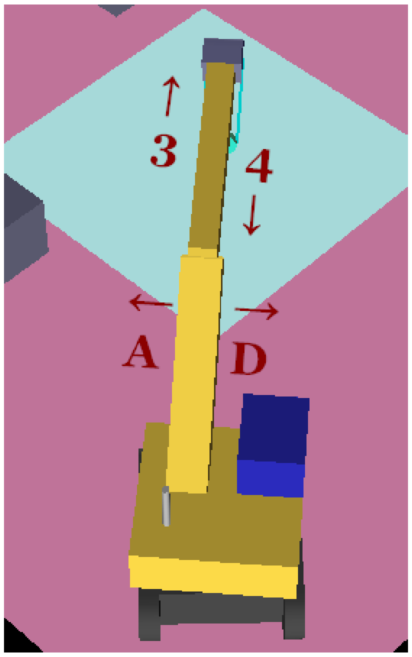


Figure 4. Controlling objects part 1

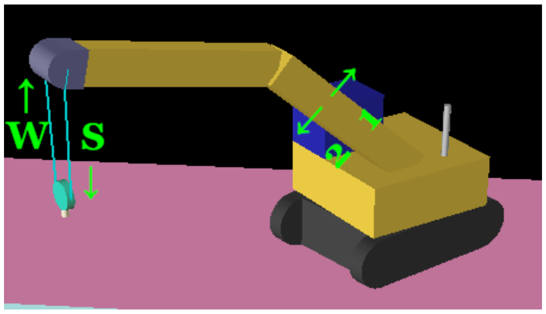


Figure 5. Controlling objects part 2

5. Window Outputs

For each stone block in the screen, we would draw it according to its status. At the beginning, no stone block is picked up, so stone blocks lie in their initial positions, as shown in Figure 1. At the mean time, the billboard window would just show *Game Start:* and the instructions. When the controlled magnet is moved to touch the top face of a stone block, the block is picked and should move with the magnet. Meanwhile, the billboard will show *...hooked...*. Next, if the picked stone block is moved to the position above the white square, which is the right position to drop, it can be dropped down to ground and the billboard will show *...placed...*. Otherwise it would be always attached to the magnet, even if the block touches the ground. When the first stone block is placed properly, we could pick up another one, moving it to the right position and dropping it down. Whenever a block is stacked, the billboard will show a score of current height of the stack.

6. Texture Mapping

In order to better differentiate whether the stone blocks are placed in right position or not, we use texture mapping to draw the stone blocks that are properly placed. Rather than use an existing picture, we generate the texture by ourselves to map it around the placed stone blocks. First we generate the texture data array in GLubyte, in which 16 color elements are included. And we cut one single square face into 16 little square parts with equal areas. Then, each color element is set for a single area and two adjacent areas are not in the same color. The original texture is shown in Figure 6. Next, with three OpenGL libraries included, namely, *gl.h*, *glu.h*, and *glut.h*. We could generate and bind texture with our target stone blocks. In configuration of the texture parameter, we use linear filtering algorithm to merge colors between one area and its adjacent areas. After filtering, the texture would look like Figure 7. Finally, when drawing stone block, by using texture coordinate, texture is mapped and added into the original color of the stone block, so it could be dark of light due to different initial color of the stone block. However, to make it look great, we decide to abandon the initial color when texture is mapped around. The resulting effect in our game is shown in Figure 8.

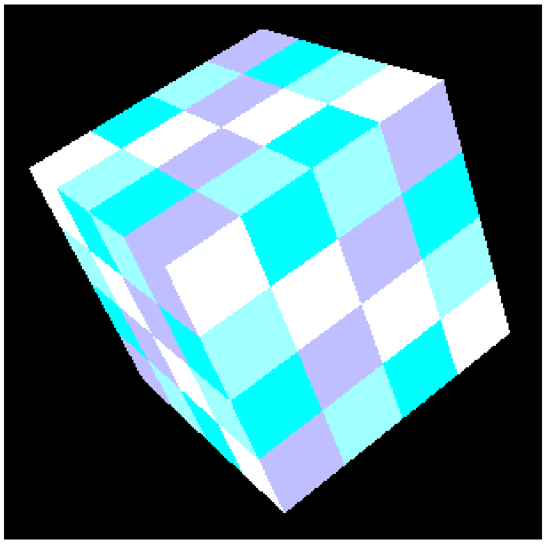


Figure 6. The initial color of texture mapping

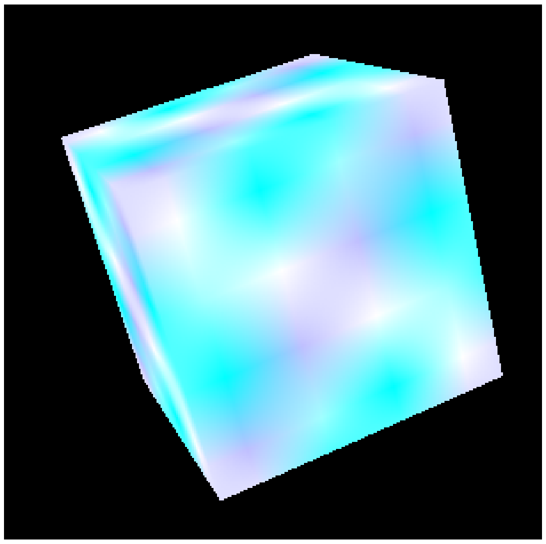


Figure 7. Texture mapping after linear filtering

7. Two Spaces in the Game

In this game, we used two spaces: one is for moving blocks (moving space) and the other one is for simulation of the stacked blocks (simulation space). The main reason for using two separate spaces is that moving blocks using a crane does not require us to know the exact collision points between the crane and blocks. The other reason is that stacked blocks requires rigid body simulation. Therefore, the stacked blocks are in the space where the simulator could access. In this game, we use Open Dynamics Engine, which is a library for simulating rigid body dynamics, to simulate the dynamics of stacked blocks.

Therefore, the collision detection schemes are different in the moving space and in the simulation space. More details would be given in later sections.

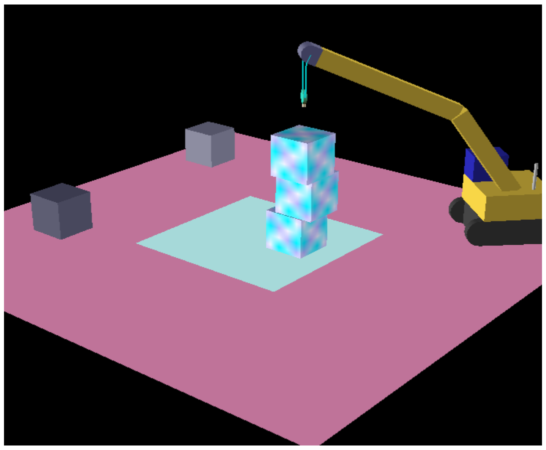


Figure 8. A snapshot of gaming

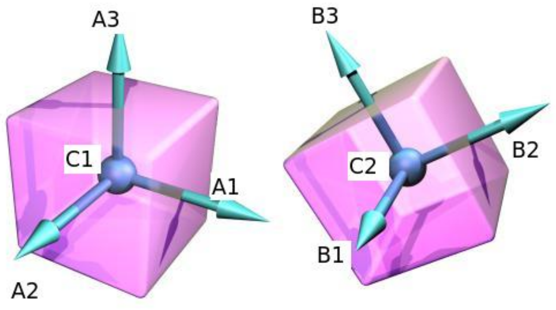


Figure 9. Two disjoint OBBs. Note that two axes of the two OBBs are not aligned [2]

8. Collision Detection in Moving Space

Since the stones and most parts in the hierarchical model, with the exception of hood-block are indeed boxes, collision detection between parts of the crane and the stones is equivalent to broad phase collision detection with OBBs. This means when the two OBBs collide, the two objects bounded by the two OBBs collide.

8.1. OBB Basics and Broad Phase Detection

OBBs are the bounding boxes whose orientation “best suits” the object they are bounding [1]. Collision detection in OBBs are more difficult than the one in AABBs because the axes of the OBBs might be aligned. For AABBs (bounding boxes aligned with the coordinate axis [1]), collision detection is simply verifying whether the range of projections of vertices overlaps with each other. However, for OBBs, the two boxes are disjoint if there exists a separating axis orthogonal to a face from either OBB or orthogonal to an edge from each [1]. We have to verify at most 15 separating axes (six from the unique edges of the two OBBs and nine from the cross products of the unique axes of box 1 and box 2).

Two OBBs are shown in Figure 9. Each box is represented by a center point C and three orthonormal axes. Each axis should be orthogonal to two of the faces. In addition, each box has two numbers a,b and c, which represent the amount of “walk” from the center to the surface in each axis.

The two OBBs in Figure 9 are disjoint. Indeed, a separating plane orthogonal to $A1$ could separate the two. In this case, the sperating axis is $A1$.

Let us consider a more general case. Suppose the separating axis we are going to verify is $\mathbf{L}$, then if the following condition is met, the OBBs are disjoint:

$$\mathbf{D} \cdot \mathbf{L} > \sum_{i=1}^{3} |a_i \mathbf{A_i} \cdot \mathbf{L}| + \sum_{i=1}^{3} |b_i \mathbf{B_i} \cdot \mathbf{L_i}| \qquad (1)$$

The main purpose of collision detection in moving space is to prevent parts of the crane penetrate into the stones. Therefore the simple collision detection algorithm described above is sufficient.

8.2. Magnet-Stone contacts

A special case we have to consider is the magnet-stone contact. When a magnet contacts with a stone not in the center region and the contact point is on the upper surface, the stone would be attached to the hood-block. Since the magnet is small, it could be considered as a particle and so finding the contact point is trivial.

9. Collision Detection in Simulation Space

When a stone is put to the top of the block stack in the center region, the stone is unattached from the magnet and added to the simulation space. The collision detection in the simulation space is handled by ODE. For each step in the simulation, collision detection is invoked. When a collision is found, the collision detection engine figures out which bodies touch each other and passes the resulting contact point information to the simulator. The simulator in turn creates contact joints between bodies. The forces between the bodies are calculated and so the position and orientation of the bodies would be computed. Rigid body simulation is described in the next section.

10. Rigid Body Simulation

Unlike particles, whose simulation takes position and linear velocity versus time into account, the rigid body simulation should also consider rotation and angular velocity versus time. Let $\mathbf{Y}(t)$ be a vector representing the state of a rigid body versus time. In fact, it is denoted as:

$$\mathbf{Y}(t) = \begin{pmatrix} x(t) \\ R(t) \\ P(t) \\ L(t) \end{pmatrix}$$

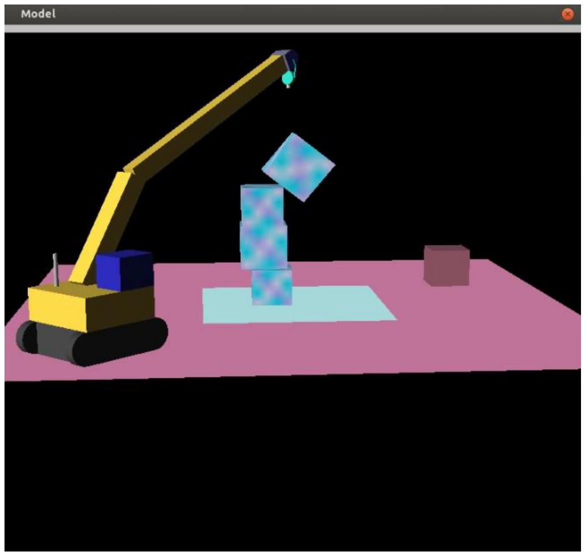
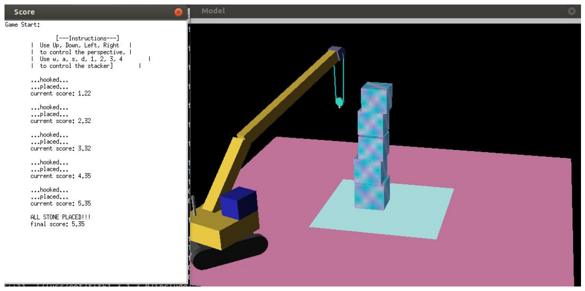


Figure 10. A stone is falling of from the stack



```
Game Start:

        [---Instructions---]
    |  Use Up, Down, Left, Right   |
    |  to control the perspective, |
    |  Use w, a, s, d, 1, 2, 3, 4        |
    |  to control the stacker]       |

    ...hooked...
    ...placed...
    current score: 1.22

    ...hooked...
    ...placed...
    current score: 2.32

    ...hooked...
    ...placed...
    current score: 3.32

    ...hooked...
    ...placed...
    current score: 4.35

    ...hooked...
    ...placed...
    current score: 5.35

    ALL STONE PLACED!!!
    final score: 5.35
```

Figure 11. All the stones are in the stack. The player wins this game

Here $x(t)$ is the location of the center of mass, $R(t)$ is the rotation matrix, $P(t)$ is the linear momentum and $L(t)$ is the angular momentum. Then the derivative is

$$\frac{d}{dt}\mathbf{Y}(t) = \begin{pmatrix} v(t) \\ \omega(t) \times R(t) \\ F(t) \\ \tau(t) \end{pmatrix}$$

Here $v(t)$ is the linear velocity of the center of mass, $\omega(t)$ is the angular velocity, $F(t)$ is the net force acted on the body, $\tau(t)$ is the torque. Using Euler's rule, $\mathbf{Y}(t_0 + \Delta t) = \mathbf{Y}(t_0) + \frac{d}{dt}\mathbf{Y}(t_0)\Delta t$. The ODE simulator uses such rule to update the states of the rigid bodies.

11. Result and Discussion

The Physics rules used in this 3d block stacker game are complicated. This is the main reason for using ODE instead of writing our own. When the center of mass of the stones in the stack are far away from each other, the stack will collapse, as shown in figure 10. When any of the stones in the stack falls, the player loses the game and the final score is given.

When the player successfully stacks all the blocks (Figure 11), the player wins this game and the final score is given. Since the size of the blocks are generated randomly and the score is the height of the stack, the player would have different scores in different games.

12. Future Work

Our project is not perfect. Therefore future improvements are necessary.

1) When the stone is attached the magnet, the orientation of the stone should change as the arm moves in some angle
2) Currently when the crane body hits the stack, the stones will not fall off. This feature should also be implemented

Other improvements could also be done. For example, we can further research on the ODE simulator and make our game more realistic. With such effort, we believe this game would be more enjoyable.

13. Conclusion

In this project we use hierarchical model to implement a crane, which could move stones and stack them in a specific region. Extensive amount of knowledge on hierarchical model, collision detection and rigid body simulation are necessary. Our result shows that our 3D block stacker game is close to a realistic block stacking. With future improvements, our game should be more enjoyable.

References

[1] A. H. Watt and A. Watt, *3D computer graphics*, vol. 2. Addison-Wesley Reading, 2000.

[2] "Ode wiki page." http://ode-wiki.org/wiki/index.php?title=Manual:_Concepts, 2011.