Physics Based Tree Chopping Simulation

CMPT 766

by
Liam Sparling
Student Number: 301464281
Kritu Patel
Student Number: 301469200

Taught By
Prof. KangKang Yin

Abstract

We implemented a physically accurate tree chopping simulation. In this simulation, a character will control an axe which they swing at the tree. The novel work done in this project is the physics interaction between the tree and the axe in addition to control of the character. When the axe intersects with the tree, it will cut a part of the tree away. The tree will fall down once enough of the tree has been cut.

Related Works

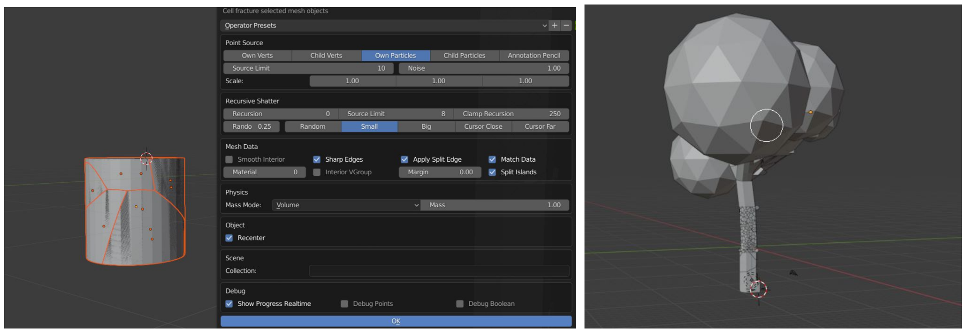
The following works from Nvidia and Embark Studios influenced our idea for this project. In 2013, Nvidia demonstrated fracturing of objects at run time [NVI13], but this was not implemented into real games until recently due to performance inefficiency. This set the baseline for run-time fracturing.
There is now an upcoming FPS game called The Finals [Stu23] which focuses on environmentally-destructive maps, where players are encouraged to use the dynamic environment to their advantage. This game allows for real time breaking of objects

Method

Tree Design

Tree Model

The 3d tree is modelled in blender for this project. To facilitate the chopping or breaking of the tree trunk for this project, a section of the tree was created using Voronoi fracturing. This method allows for the sectioned pieces to be random while allowing us to define the level of granularity of the pieces of the tree. For our project, we decided to discretize the tree into approximately 500 pieces.


Cell fracture selected mesh objects
Operator Presets

Point Source
Own Verts | Child Verts | [✓ Own Particles] | Child Particles | Annotation Pencil
Source Limit 10 | Noise 1.00
Scale: 1.00 1.00 1.00

Recursive Shatter
Recursion 0 | Source Limit 8 | Clamp Recursion 250
Rando 0.25 | Random | [✓ Small] | Big | Cursor Close | Cursor Far

Mesh Data
Smooth Interior | ✓ Sharp Edges | ✓ Apply Split Edge | ✓ Match Data
Material 0 | Interior VGroup | Margin 0.00 | ✓ Split Islands

Physics
Mass Mode: Volume | Mass 1.00

Object
✓ Recenter

Scene
Collection:

Debug
✓ Show Progress Realtime | Debug Points | Debug Boolean

OK

Figure 1: Tree Model with fractured bark

In order to simplify the project, we only fracture the middle section of the tree. Both the top and bottom sections are split from the tree before discretization. This approach allows us to keep the model as simple as possible.

Tree Physics

We calculate collisions with the tree using Unity's collision logic. When an object hits the tree, a radius is proportionally calculated based on the force or velocity of the object assuming the force surpasses a minimum breaking value stored in a variable called *strength*. Using this radius and the point of intersection of the initial object, we create a sphere. Objects inside the sphere are considered to be broken off from the tree, and therefore detach from the tree and delete themselves after 3 seconds. Additionally, a small force is applied to the broken off pieces so that they remove, or fly away from the tree, as if they are being launched from the force of the collision.

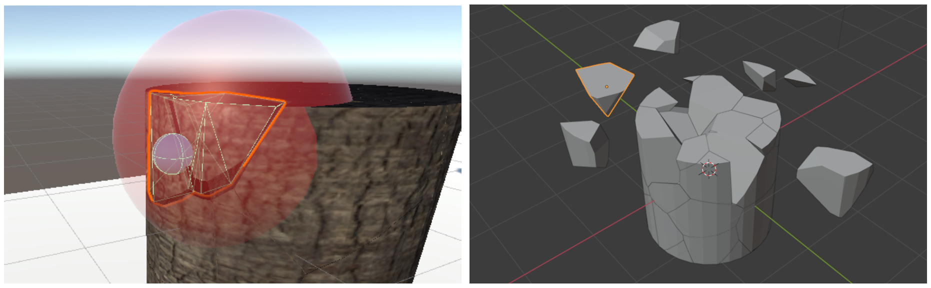


Figure 2: Impact Calculations

Tree Falling

The tree falling logic is based on a ratio of the remaining pieces of the tree compared to the initial pieces and whether it crosses the set threshold of chipping for the tree to fall. For example, if the threshold was 90, then once 90% or less of the tree remained, the tree would fall. Furthermore, the tree trunk is divided into sections based on height and if a enough wood for any of those sections is chipped away the tree falls.

Floating Pieces

While breaking off pieces of the tree, there can be situations where floating pieces, seemingly unattached to any other piece, remain. In order to help alleviate this problem, we implemented an algorithm which would check if any piece is currently touching the floating piece. This algorithm finds the minimum and maximum bounds of the mesh for each piece and uses Unity's bounding box method to find all pieces encapsulated by that bounding box. The piece is considered floating if no pieces are found. We used only half the size of the mesh bounding box as we found that Unity would collide with objects that were simply close to, but not actually touching, the piece we were considering.

Player Control

Lumberjack Character

The character is a simplified humanoid in this project. For our application, we are only considering that the right arm can be controlled. The rest of the character will simply have a right arm which reacts to the wrist moving using inverse kinematics. This approach was implemented using Unity Animation Rigging and allows for two control points, a target and a hint value. The target value influences the position of the wrist and the hint value influences the position of the elbow. The target value allows for 3 position and 3 Euler angle control points to position and rotate the wrist. For the hint value, only the position of the value is relevant, and the elbow will attempt to point in the direction of the hint value. With this approach, only the arm, which holds the axe, is controllable. This allows for 6 degrees of freedom, or control points, 3 for x,y,z position and 3 for the x,y,z rotational angles for the right wrist. The arm of the character is bound in space so you can only move the wrist in space within a certain area to the right of the tree.

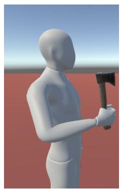


Figure 3: LumberJack Holding Axe

Axe Chopping

For the axe, we developed a simplified physics model to ensure it properly interacts with the tree. In a real life situation, you would ideally want the blade of the axe to be parallel to your force vector. In order to enforce that this type of motion was used to hit the tree, we added a chopping animation to damage the tree. To create this chopping animation, we first find the vector facing along the direction of the axe, which is the same direction as the hand is facing. This vector is used to linearly interpolate (LERP) the axe along the vector's direction for one unit distance in Unity space. A distance of one was chosen because it aligned well with the max range of the human arm.

As the axe is held by the character's hand, it is similarly moved using the target value from the animation rigging constraint. The speed of the LERP is controlled by the user and affects how much force is applied to the tree. At its slowest, the LERP will take, up to, 1 second to finish, and send a force value of 1. At its highest speed, the LERP will take 0.1 seconds to finish, and send a force value of 10 to the tree. Between those values, the force is linear. Additionally, the Axe will immediately stop if it hits the tree.

Manual Control

As mentioned above, manual control of the character was implemented using key bindings. Key bindings were used to control the axe's position and angle from the tree. The following key bindings were used for position:

- W (further) and S (closer): Controls the depth of the axe (or Z axis)
- A (left) and D (right): Controls the horizontal movement of the axe (or X axis)
- Q (lower) and E (higher): Controls the height of the axe (or Y axis)

For angular movement, the following key bindings were used:

- I and K: Controls the roll
- U and J: Controls the yaw
- O and L: Controls the pitch

In order to activate the axe swing, we use the space-bar. As the space-bar is held, the power level of the axe raises. When the space-bar is released, the axe swings, with speed according to the current power level.

As this is not a fully developed simulation, there are some freedoms the user can perform that are not physically accurate. The player can move the axe through the tree manually with the controls. This allows the player to effectively break the simulation. As the player moves through the tree, the axe will not deal damage to the tree, only if the user uses the axe chopping feature will it damage the tree. In this case, the user is not supposed to chop the tree unless their axe is in a reasonable location.

GUI

To allow the user to better understand the current power level of their axe, we added a slider bar on the top right corner of the screen to display the current power of the axe. This bar gives the user a general idea of how much force they are applying.

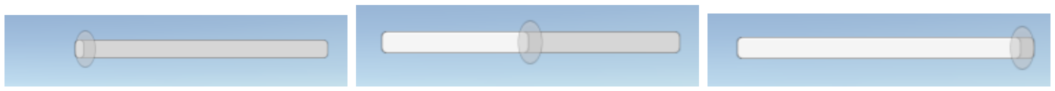


Figure 4: Power GUI Element (Lowest Power, Half Power, Max Power (Left to Right))

Challenges

Sometimes we have pieces which are floating in space after a few pieces of wood have been cut away. We implemented an algorithm to help with this but it does not always work. One of the reasons for this behaviour is that during the discretization in blender, it can create pieces which are invisible because the mesh is extremely small so unity can not render it. In this case, the algorithm still sees these pieces as present and therefore prevents pieces, which are seemingly floating in the air, from falling, because they are actually attached to these pieces. Another issue with the floating pieces algorithm is that it does not account for sections which are floating in the air. As a section has multiple pieces, it considers it to be attached and therefore does not fall.

Work Distribution

- Kritu - Tree Modeling and Fracturing, Tree Cutting Setup, Tree Falling Baseline, Launching Pieces
- Liam - Character Setup with Axe, Tree Cutting Modifications, Tree Falling Sections, Floating Pieces Handling

References

[NVI13] NVIDIA. *Real Time Dynamic Fracture with Volumetric Approximate Convex Decompositions:* Youtube. 2013. URL: https://youtu.be/eB2iBY-HjYU.

[Stu23] Embark Studios. *The Finals*. Embark Studios. 2023. URL: https://www.reachthefinals.com/.