Thomas Lefley
Girton College
tl364

Computer Science Part II Project Proposal

Mesh to voxel transformation for optimised physics-based interactions

14/10/2014

Introduction and Description of the Work

Current trends in digital gaming are towards sandbox titles, where the player is encouraged to shape and manipulate the world, often by placing and breaking blocks[1]. This is achieved by storing the state of the world as a three dimensional array of block identifiers, usually batched into 'chunks' which are loaded and unloaded as the player approaches them. While this approach has been optimised for memory and CPU load, it presents several limitations with regards to player experience. Players are confined to placing and breaking blocks in a grid due to the underlying data structure and so are unable to create landscapes without object placement looking unnatural and formulaic. Furthermore, the data structure only allows location and type to be stored for each block, with location being derived from the block position within the array. This means that blocks cannot interact physically with the world and thus the world is very static.

Solving these issues will be the goal of my project. My proposed solution is to create a hybrid between traditional polygonal mesh and sandbox engines. Game objects will exist primarily as mesh objects, however mesh to volumetric conversions will occur at appropriate physical cues. The updated meshes will then be reformed after the interaction has taken place. By treating objects as volumetric when interacted with (via physics or the player), the creative freedom of sandbox engines will be maintained, with all objects having the capacity to be reshaped. Objects existing as polygonal meshes for the remainder of the time will allow for both non discrete object placement[2] and physically based movement.

One main project focus will be physically based object deformation. As objects will be volumetric on collision, I will use this to simulate accurate reactions. Treating each voxel making up the object's volume as a unit of mass and forming rigid bonds between these units, I will propagate collision information through the bonds from the point of collision and calculate which should break or be otherwise changed. This will allow objects to be warped or broken apart depending on parameters such as the object's strength, brittleness or malleability as well as the force and position of impact. All of this will be calculated internally before any required game objects are created.

For example, if a bullet hits a brick wall then you would expect units of mass to break off and fall to the ground as rubble. At the moment of impact, the wall will not disappear from the game world to be replaced by $x \times y \times z$ unit cubes bonded together, some of which breaking off before the updated wall reappears. Rather, the wall will compute its internal structure of voxels, work out which bonds should break, form new objects from the severed mass, apply vector quantities such as velocity to these and then update its own mesh using the remaining mass. This approach is taken because game objects are expensive and therefore to be optimal we should be lazy with their creation.

Furthermore, computation of voxel structure should be both patch based and limited to impacts above a threshold. This is because not all impacts will result in deformation of the whole object, or any at all.[3] A block of wood dropped to the floor from a height of 1 inch will undergo no deformation. A steel block dropped into a field from a height of 6 feet would not require voxelization of the entire field, only a radius around the point of landing.

[1]See Minecraft by Mojang

[2]That is, objects do not have to remain snapped to the underlying array

[3]As dictated by the object in question's physical parameters

Starting Point

I will be working using the Unity3D physics and game engine, with which I have prior experience. The engine will provide general features such as rendering, physics and game object management which I will expand upon to achieve my more specific goals. As Unity3D is a widely used commercial engine, its use will also allow for comparable evaluation against scenes implemented using only the base features.

Substance and Structure of the Project

This project will involve writing a library for an existing mesh physics and rendering engine, Unity3D, which will perform mesh to voxel transitions efficiently at runtime, as well as exhibit the correct physical response amongst the individual voxels of the volumetric object.

The project has the following main sections:

1. Runtime patch voxelization of polygonal meshes.
2. Mesh updates and formation from volumetric structural changes. Presumably using an approach similar to the marching cubes algorithm.
3. Physically based deformation of volumetric objects.
4. Player based manipulation of volumetric objects.
5. Writing the dissertation.

If time allows, two further areas will be explored:

1. **Texture mapping.** Thus far I have only detailed physical deformation, with each object and any further objects formed via their destruction being uniform in texture[4]. However, an ideal system would allow for more complex textures and deform these to mirror the physical mesh destruction.
2. **Physics at rest.** The main focus of the project is physics caused by interaction such as two objects colliding. A further goal would be modelling the internal physics of resting objects. Take for example a wooden beam fixed at one end with a weight at the other, if this weight is too heavy, the beam should break at the correct point. This simulation is trickier as the cue for when to voxelize and resolve is not as obvious.

[4]Likely a solid colour

Success Criteria

The following should be achieved:

- Implement and demonstrate runtime conversion between mesh and volumetric objects.
- Implement and demonstrate different and expected physical responses based on collision and material variations.
- Implement and demonstrate player modification of game objects.

The following are evaluation criteria:

- A scene with a reasonable number of objects should run well[5] on a computer with moderate specifications.
  - Comparable scenes built using both conventional voxel only and mesh only approaches will be used for efficiency evaluation.
  - My own computer will provide a high specification test case, the MCS machines found in the Intel Lab at the William Gates Building will represent a mid range specification.
- Conversion between mesh and volumetric objects should not be noticeable to the player.
  - There should be no obvious slowdown or lag when a conversion occurs.
  - The point at which any mass breaks from the parent object and the parent object mesh updates should not be jarring visually.

The following are stretch goals to be achieved if time allows:

- Improved texture capabilities.
- Simulating internal physics at rest.

My demonstration will consist of two parts. I will present the project from both a technical and end user standpoint. In the former I will show slowed down and annotated mesh to voxel conversions and subsequent physical reactions whereas the latter will consist of how a player might interact with a scene built with this system.

[5]I would define ‘well’ to mean a framerate above 30FPS, with CPU and memory loads no more than 1.5$\times$ those found when using standard techniques

Timetable and Milestones

Sprint 1 (24/10/14–7/11/14)

Study of algorithms involved in mesh to voxel conversions and the inverse. Basic implementation of these algorithms within the engine.

Milestones: Some example code, which will probably not be used in the final project, and some example scenes demonstrating conversion.

Sprint 2 (7/11/14–21/11/14)

Refactor the algorithms to be internal computation with no physical representation of voxels in the game world. Reimplement voxelization to be patch based.

Milestones: Working patch voxelization. Underlying data structure for voxels and inter-voxel bonds near finalised.

Sprint 3 (28/11/14–5/12/14)

Hook the procedures to the correct physical cues. Optimise these algorithms

Milestone: Algorithms which will likely be in their final form. Examples of conversion occurring on cue.

Sprint 4 (5/12/14–16/1/15)

Review code and progress over the Christmas break. Ensure everything is well documented, including source and evaluation of current progress.

Sprint 5 (16/1/15–30/1/15)

Write initial chapters of the dissertation. Write progress report

Milestones: Preparation chapter of Dissertation complete. Submission of progress report

Sprint 6 (30/1/15–13/2/15)

Begin to work on physically based destruction algorithms for the splitting apart of objects, including transferral of force only any ‘splinter’ objects. Define parameters which will impact physical reactions.

Milestones: Ability to define physical properties for game objects. Data confirming some level of impact propagation through voxel bonds.

Sprint 7 (13/2/15–27/2/15)

Integrate voxel to mesh algorithm with the return from physically based destruction.

Milestones: Able to see object split apart as the result of an impact of the correct magnitude.

Sprint 8 (27/2/15–13/3/15)

Implement volumetric deformation of objects on physical cues.

Milestones: Examples of malleable objects bending under force.

Sprint 9 (13/3/15–27/3/15)

Implementation of player alteration to objects.

Milestones: Example scene where playing is able to manipulate objects via physical and non-physical means.

Sprint 10 (27/3/15–10/4/15)

Code review and code freeze. All success criteria should be implemented by this point. No new features will be written during this period, only bug fixes and optimisations.
Further dissertation progress.

Sprint 11 (10/4/15–24/4/15)

Creation of comparable scenes using pure voxel and mesh methods for profiling.
Profiling against these scenes as well as on computers with varying specifications.

Sprint 12 (24/4/15–8/5/15)

Finish dissertation, preparing graphs for insertion. Review whole project, check the dissertation, and spend a final few days on whatever is in greatest need of attention.

Final Sprint (8/5/15–15/5/15)

Final proofreading and submission of dissertation.

Milestone: Submission of Dissertation.

Resources Required

I will be working with a free license of Unity3D installed on my own computer with the following specifications:

- 3.40GHz i5-4670k
- 8GB RAM
- 400GB remaining SSD storage
- 750GB remaining HDD storage
- AMD Radeon HD 7950 GPU
  - 3GB GDDR5 Memory

I accept full responsibility for this machine and I have made contingency plans to protect myself against hardware and/or software failure.

All source code will be be backed up and version controlled using Git with GitHub. Documentation, as well as any further resources, will be created and stored in a Microsoft SkyDrive cloud folder with regular backups to Google Drive as well as my own hard disk drive.