A Collaborative Game Development Tool

Brian Thorpe[#1]

Computer Science Department, University of Colorado at Colorado Springs
1420 Austin Bluffs Parkway, Colorado Springs, Colorado, USA
[1]bthorpe@uccs.edu

***Abstract*— This document describes an approach to developing a collaborative game development tool. The paper describes an approach for the tool, the design for the tool, and an implementation of such a tool.**

***Keywords*— Game Development, Collaborative, Networking, Tool, Terrain, Models.**

I. Introduction

Game development requires a significant amount of input from a variety of roles. Artists, programmers, game designers, story writers, all are involved in the creation of the game. There could be several different artists producing content for a scene in a game, one may be producing textures for buildings, and other textures for the ground. A collaborative tool is needed to facilitate the development when these assets are being placed into a game. This would allow for artists, and game designers to have the ability look at the same game scene at the same time and discuss modifications to provide a user the best game play experience possible.

A collaborative game development tool would be a composition of other game development that share the data between clients through a server. Some of the tools included in a collaborative game development tool could be a terrain editor, an asset placement tool, lighting effect tools, character placement, etc.

The paper describes an approach to developing one such collaborative game development tool, a terrain editor. The terrain editor is synchronized with a server. When a client connects it receives the current server data, and the current server state. Once synchronized the user can begin looking at the current game content, and begin creating new terrain content.

The paper discusses the design, implementation, and future enhancements of a collaborative terrain editor.

II. Collaborative Terrain Editor

The Collaborative terrain editor is composed of both a client and a server. The client and server work on a standard client server model. Clients connect to the server, and the server communicates back to the other clients.

A. The Client

The client is composed of game data, a game engine for rendering the game data, a user interface for modifying the data, and a client for connecting and communicating with the server.

1) Client: The client will communicate with the server by sending and receiving messages. The messages are a set of defined packet types which will contain the different operations necessary to synchronize and edit terrain.

2) Client Terrain: The client terrain is composed of several components. The elevation data for the terrain which is shared with other clients and the server. The terrain geometry, for rendering the terrain mesh. Mesh triangle data for performing picking operations and collision detection.

The elevation data is being kept synchronized with the server. This data is updated when edit messages are received which further updates the other client data components.

3) Game Engine: The game engine is responsible for rendering the client terrain to the user. The game engine in his case is a deferred shading graphics engine built to render the terrain and texture it using a height normal technique.

4) User Interface: The user interface provides the user with the ability to select the editing tools, and their parameters. The user needs to be able to raise, lower, flatten, and smooth terrain as well as add additional pieces of terrain. The user needs to be able to select three different parameters for the edit, the inner radius of the edit, the outer radius of the edit, and the amount of power applied for the edit.

B. The Server

The server is composed of only the game data, and the server for connecting an communicating with the clients.

1) Server: The server can handle chat messages, edit commands, and synchronization data. The server processes this incoming data and forwards it to the client.

2) Server Terrain: The server terrain is only the elevation data. The server uses the same processing commands as the clients to apply edit data requests.

3) Server State: The server state holds all of the edit requests made by clients since the last save. This state allows for a client to join in during an editing session, acquire the current baseline file data and then synchronize to the current state in the server.

III. Collaborative Terrain Editor Process Design

The synchronization and editing process has three major steps. The first step is to synchronize the files with the server, this is referred to as file syncronization. The second step is to synchronize to the state of the server, this is referred to as

state synchronization. The final step is the editing step, this is the state at which clients send and receive updates actively seeing changes made by other clients.

A. File Synchronization

The server maintains a list of files it is serving to clients. The list of files is associated with an MD5 hash of the file. When a client connects to the server, the client sends the MD5 hash of the client's local content. Any mismatches or missing files are sent back to the client. Once the client has the correct files it can be considered synchronized to the files.

B. State Synchronization

Once a client is synchronized to the files on the server, the client must synchronize to the active state of the server. The server's active state is a set of modifications to the data since the last save to the files. The server transmits any modifications back to the client for processing, once processed the client can enter the editing state.

C. Editing

Once the client is in the edit state, the client actively listens for modifications by other clients transmitted by the server. The client can also make edits to the terrain data and the server listens and forwards the requests to the other clients.

IV. Collaborative Terrain Editor Implementation

To test the design a client and server were implemented in C#. The client makes use of a preexisting rendering engine developed by the author, and the Lidgren C# Networking Library. The server uses only the Lidgren C# Networking Library.

The large portion of the logic is located in the design of the terrain, and terrain editing. The rest of the logic is in the client-server communication.

A. Terrain

Terrain is the ground representation in some games see Fig. 1. In this particular implementation terrain is a set of grids of vertexes that can be assigned to a location on the x,z plane. The y values account for the height data on this plane.

Terrain pieces are 129x129 vertexes of height data, called a terrain block. The server is responsible for maintaining the height data for each Terrain Block. The server is responsible for updating the height data, and the normals associated with it. Whereas the client is responsible for updating this data to match the server but also updating the other data structures to display and interact with the terrain correctly. In this particular terrain implementation vertex morphing information is stored as well to facilitate smooth switching from different levels of detail.

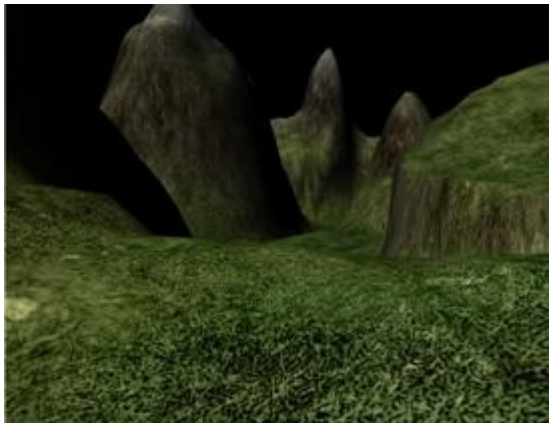

Fig. 1 Example of game terrain from client.

1) Terrain Elevation Data: Applies to both server and client. Terrain elevation data is the set of data holding only the terrain elevation data that is being stored on the server and synchronized with the clients. This is the dataset that the client and server both store and manipulate to change the terrain. Terrain Elevation Data stores the same number of vertices as a Terrain Block. For this implementation, Terrain Elevation Data is composed of two Vector4 representations which is a grouping of 4 floats for a total of 8 floating point numbers. The first 3 (x,y,z) components of the first Vector4 correspond to the normal of the vertex, the 4th (w) component corresponds to the elevation. The first 3 (x,y,z) components of the second Vector4 correspond to the morphed normal of the vertex, the 4th (w) component corresponds to the morphed elevation. The second Vector4 is the extra information to support vertex morphing.

The Terrain Elevation data is stored in a binary format on the server and transmitted to clients upon connecting. The format when saved is the output of both Vector4s x,y,z,w,x,y,z,w and can be read in by reading singles from a byte stream.

2) Terrain Blocks: This applies to the client only. A terrain block consists two pieces. The Base Terrain Block, and the Terrain Elevation Block. The Base Terrain Block is set of x,z values corresponding to the 129x129 vertexes for a Terrain Block stored in a Vertex Buffer. The Terrain Elevation Block is only the height data in this case the Terrain Elevation Data, moved into a Vertex Buffer. This design is to reduce the memory footprint of terrain by removing the x,z data that is stored with the elevation and reused for each Terrain Elevation Block.

129x129 vertexes was chosen for several performance enhancing properties. 129X129 vertexes allows for multiple levels of detail by removing every other vertex. This can be done multiple times since the number of quads formed by the vertexes is a power of 2. 129x129 is also the largest number of vertexes alllowing for the level of detail processing and enabling indexing using a short two byte value.

When the Terrain Elevation data is edited the Terrain Elevation block must be updated as well. This requires

locking the vertex buffer on the graphics card and writing the new terrain elevation data to the vertex buffer.

3) Terrain Patches: This applies to client only. A Terrain Patch is a subset of the larger terrain. The patches only serve as a way of building a subset of a terrain for rendering a smaller portion or multiple pieces of the terrain with different levels of detail. Each patch contains a set of index data which corresponds to the triangle formation scheme for the specified level of detail. This does not need to be updated when the terrain is edited.

4) Terrain Physics: The terrain is represented in the games physics engine as a set of triangle data. This data is used to perform the picking operations on the terrain to determine the cursors location in the 3D coordinate system. The triangle height data is updated when the terrain is edited.

5) Terrain Edit Parameters: The terrain edit supports editing height values with a round paint brush. Two parameters control the inner and outer radius of the brush as s shown in Fig. 2. The inner radius edits terrain with the maximum power. The outer radius edits the terrain with a linear degradation towards the outside. The power parameter controls how strong an effect the edit has on the terrain. The parameters are controlled by the client using the GUI show in Fig. 3.

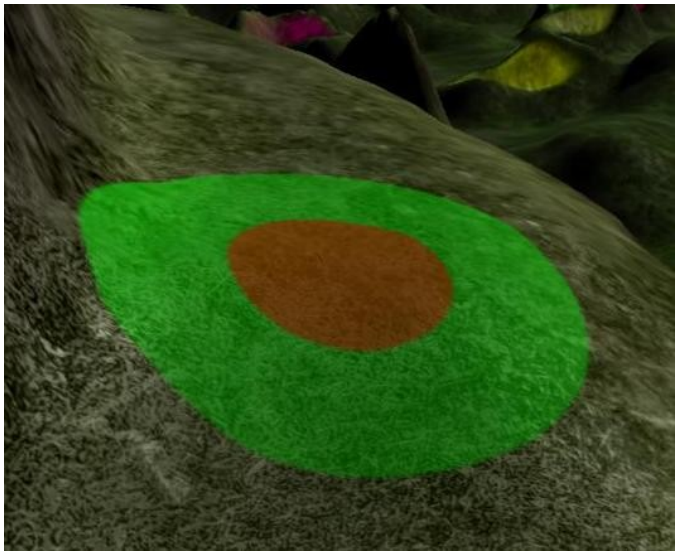

Fig. 2 Example of terrain edit brush with inner radius (red) and outer radius(green).

Fig. 3 Example of of terrain edit parameters.

1) Terrain Edit Strategies: The terrain can be edited using four different edit strategies. The raise strategy raises the terrain within the cursor by the amount given by power, degrading linearly in the outer radius to 0. The lower strategy lowers the terrain within the cursor by the amount given by power degrading linearly in the outer radius to 0. The flatten strategy move terrain up or down depending on the difference to the cursors center point by the difference multiplied by the power this degrades towards 0 linearly in the outer radius. The smooth strategy averages the elevation of the neighboring four vertexes and moves the vertexes towards the average height difference multiplied by the power; this degrades towards 0 linearly in the outer radius. The four strategies are shown in Fig. 4.

Fig. 4 Example of terrain strategies control with raise selected.

B. *Packet Types*

The client and server communicate using pre defined packet type messages. The packet types are given in Table I, Packet Types.

TABLE I
PACKET TYPES

Packet Type	**Packet Data**
Login	Type-Byte Username-String
Logout	Type-Byte
Chat	Type-Byte Message-String
AssetRequest	Type-Byte FileName|MD5|FileName|MD5|...-String *List of FileNames and MD5s split with '|'
AssetTransferStart	Type-Byte Name-String Size-Integer
AssetTransferAck	Type-Byte
AssetTransferProcess Complete	Type-Byte BufferPosition-Integer Length-Integer Data-Byte[Length]
SyncStart	Type-Byte
SyncComplete	Type-Byte
TerrainCreate	Type-Byte PositionX-Single PositionY-Single
TerrainEdit	Type-Byte EditType-Byte NumberOfEdits-Integer TargetX-Single TargetY-Single TargetZ-Single RadiusX-Single RadiusY-Single Power-Single *TargetX-Power repeats for NumberOfEdits

Login is a packet sent containing a single string Username which displays in both the server and client it is used to identify users in chat conversations. The user name is also

announced to the clients when connecting and logging into the server.

Logout is a packet sent to the clients when a user disconnects from the server. It notifies all of the clients that a user has left.

Chat is used by the client to communicate to other users when working on the data. The Server appends the username to the front of the message before sending it to all of the clients.

AssetRequest is used by the client to send a list of file names and files hashes to the server for comparison. When the server receives the file list and files hashes it compares the files determines which file are out of date or missing and sends them to the server.

AssetTransferStart is used by the server to signal the client it is about to start sending a file. This allows the client to open a file for writing the server data.

AssetTransfer contains the position in the file buffer, the length of data, and the data to be written to the file. These are generated by the server and are a maximum number of bytes and continues sending these packets until the client has each piece of the file.

AssetTransferProcessComplete is sent by the server when the server has no more files to send to the client. The client can then load the data files it has and begin the client visualization.

SyncStart is sent by the client to the server to inform the server it has loaded the file data and is ready to synchronize the data with the server.

SyncStop is sent by the server to the client to inform the client it is now synchronized and ready to edit.

TerrainCreate is a message sent by a client to the server requesting a new block of terrain be added. The x,z positions are in grid coordinates not actual coordinates.

TerrainEdit is a message sent by a client to the server requesting a specific edit. These messages are batches by edit type and sent to the server every .1 seconds. The coordinates sent in this message are in game coordinates.

C. File Manager

The file manager for both the client and the server store the list of actives files and their hashes. This is used for loading the data and making comparisons on the server. The File Manager is also responsible for receiving the segments of files and building the files on the client as the data comes from the server.

D. Client

The client is composed of all of the terrain components, the file manager, a GUI to interface to the terrain, and chat form.

The user interface for the Client has three major sections. The game window which allows for terrain edits to be made. The chat window which allows client communication. The parameters window which allows the user to change the edit type and brush parameters.

When a user has the mouse over the game window the cursor follows the mouse pointer. When the user clicks the left mouse buttons edits are applied. The user navigates using w,a,s,d for forward, backward, left, right movement respectively. Forwards movement is in the direction of the camera. The user can move up and down using q up, and z down.

When a user makes an edit by pressing on the game window, first a ray is projected into 3d space via the cameras directional vector and the screen coordinates clicked on. This ray is intersected against the physics triangle data to return the resulting intersection point. This intersection point is used to determine where the center of the ground cursor is and where the edits are applied. The parameters are passed to one of the four edit strategies and the edit is performed. The target and parameters are then sent to the server where this edit is also applied. The only time actual vertex data is sent to the client is on the file synchronization step, the rest is done by sending edit information which is then calculated on the server and client data sets.

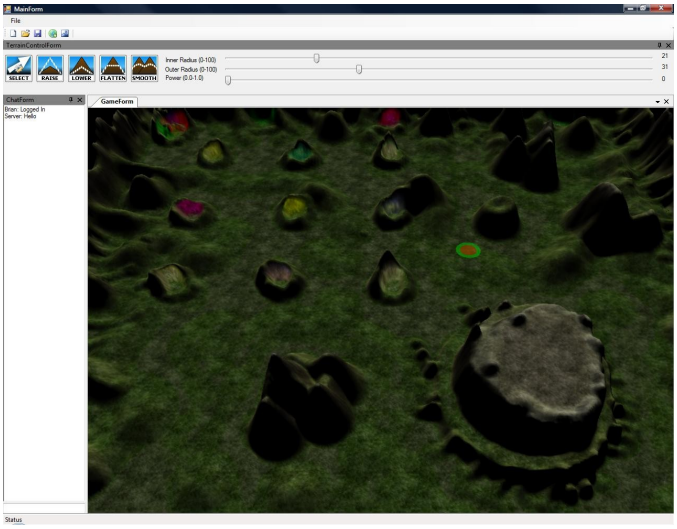

Fig. 5 Screen Shot from the Client Editor

E. Server

The server is composed of four windows, the assets form which displays what files are being shared for synchronization, the asset transfer window which displays active file transfers, the users form which displays connected users and the chat form which displays any chat messages.

The server only displays information to the users and allows a user to save. The server provides no other features to the user other than displaying information.

When the server receives an edit data packet the edit is applied to the servers terrain dataset and then the edit packet is forwarded to each of the clients.

The server and client both use Lidgren C# Networking Library for communication. Lidgren handles all of the connection setup and tear down, as well as sending the data buffers and resolving when packets are lost, and how to respond. Lidgren is based on UDP, and provides four transfer modes on a number of channels. Reliable Ordered is used for the server and client to guarantee the edits are kept in sync with the server.

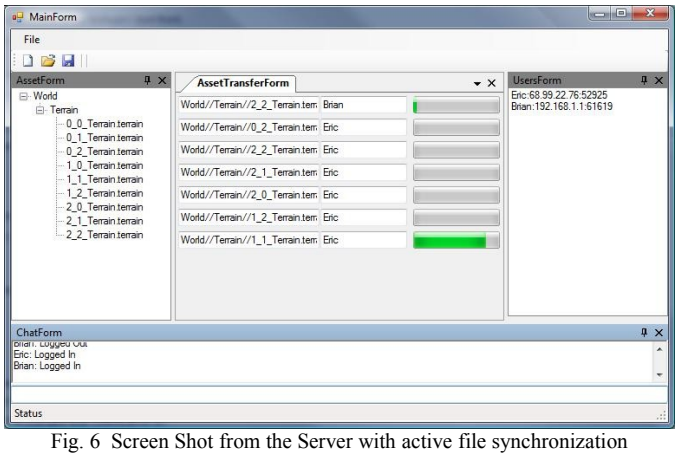

Fig. 6 Screen Shot from the Server with active file synchronization

V. Future Work

This project only looked into creating a single collaborative tool which could be part of a larger collaborative game editor. This tool provided simple but effective means for manipulating terrain data across a network. The next step would be to integrate additional tools to work with the terrain and the collaborative setup.

The first additional tool that would be easy to integrate and add value to a collaborative environment would be an asset placement tool. This tool would allow a user to import a 3D model and manipulate its position and orientation. This could be used to create forests or castles, see Fig. 7.


Fig. 7 Screen Shot from client with proposed 3D Model Manipulation

VI. Conclusions

After implementing the collaborative terrain editor. The software was tested with three users working on a set of terrains. One user was local, and hosting the server, while one was located in state, and the other out of state. All three users were able to communicate effectively while developing the terrain. The tool showed great potential for becoming a valuable game development asset.

With additional tools incorporated into this collaborative editor the tool could be used to produce game content.

References

[1] (2009) The Lidgren-Networking Library website. [Online]. Available: http://code.google.com/p/lidgren-network/