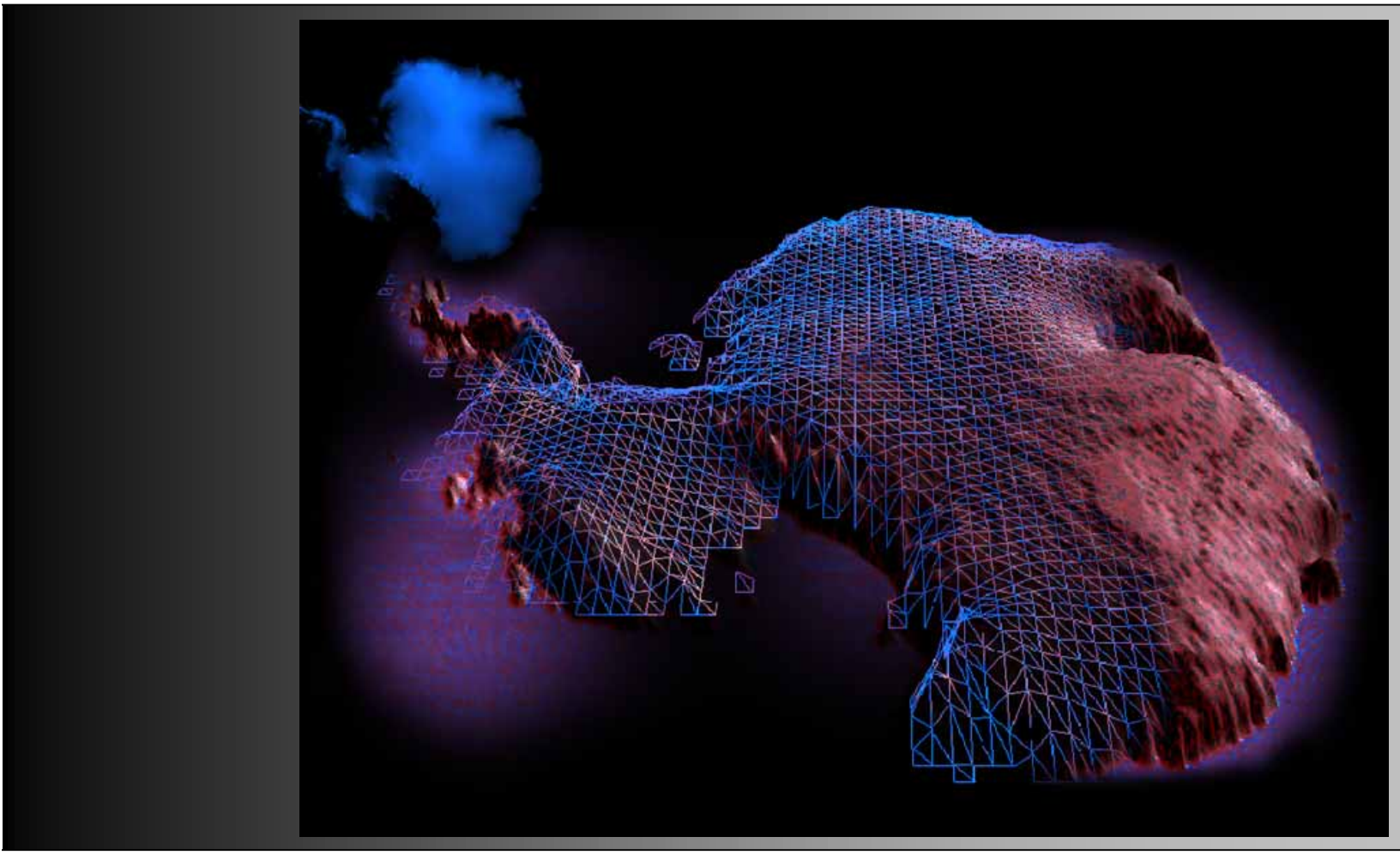
TerraForm3D

Terrain Modeling Software Design



Trent D'Hooge

Heather Jeffcott

Craig Post

Deborah Lee Soltesz

20 February 2000

Table of Contents

Table of Figures

Section 1: Overview

Contents

Figures

QuickView

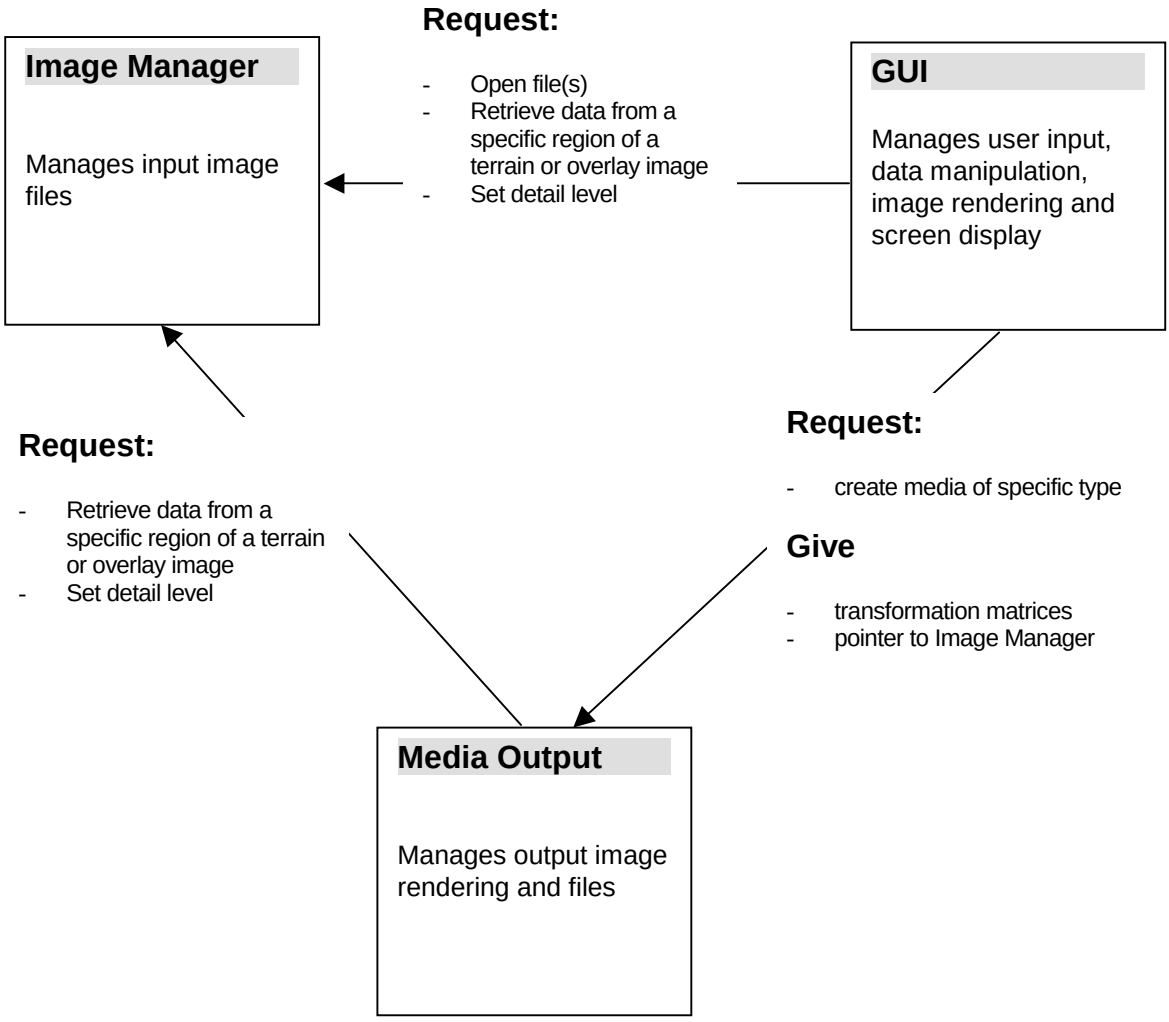
TerraForm3D is terrain modeling visualization tool that allows users to interact with a terrain model, and export the terrain model into a variety of standard products. The software is broken into three main packages:

1. TerraGUI: handles on-screen user interface display and terrain rendering, and user interface objects and events.
2. Image Manager: handles input of data from terrain and imagery files
3. Media Output: handles output of digital visualization products

TerraForm 3D

Package Interface View

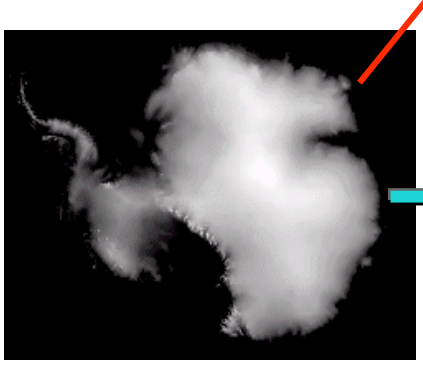
Figure 1-1: Shows high-level internal interactions and data flow among packages

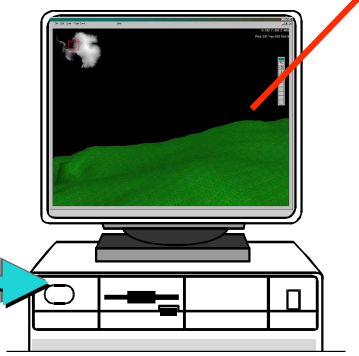


High Level Design Component Interaction Overview

Data Manipulation: The DEM is represented as a 3D terrain in the application window, allowing the user to perform real-time flybys, create on-screen views of the data, and save position and attitude information as a flight track or camera view. Screen output depends on the ***3D Engine*** for screen rendering and hardware acceleration.

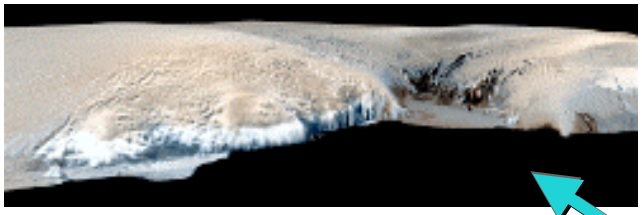
Input Data: Digital Elevation Model (DEM). Elevation values in a 2-dimensional array are represented visually as bright tones for higher elevations and dark tones for lower elevations.

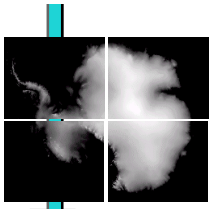


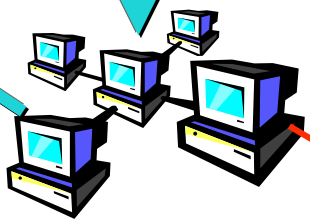


Product Generation: Elevation data and user -defined parameters are sent to file rendering component

Perspective View Image







Parallel Computations: High quality perspective views, movie sequences, and other computationally intensive functions are performed in parallel to reduce rendering time

Figure 1-2: Interaction of Components in the Terrain Modeling Package

Breakdown of Tasks and Data Management

Generalized Tasks by Package

Package	***Image Manager***	***GUI***	***Media Output***
Purpose	Manage image data files – instantiates the correct image format class based on file extension and header information	Serve as the interface between the human user and terrain data, allowing the user to visualize and manipulate the data	Create media output files using 3D display data and image data files
Tasks	Set detail level Set terrain file Set image overlay file Get terrain object Get image overlay object Get terrain data in vertex form	Handle keyboard events Handle mouse events Handle GLUI (window widget) events Display 3D rendered data	Create Perspective View Create Movie Frames Create VRML file

Data Encapsulation by Package

Package	***Image Manager***	***GUI***	***Media Output***
Purpose	Manage image data files – instantiates the correct image format class based on file extension and header information	Serve as the interface between the human user and terrain data, allowing the user to visualize and manipulate the data	Create media output files using 3D display data and image data files
Data	Terrain image Overlay image Detail level	Image Manager User input: keystrokes, mouse events 3D display data Color-code (elevation colorizing)	Image Manager Transformation matrix

Section 2: TerraGUI User Interface Design

Contents

Figures

QuickView

TerraGUI serves as the user interface for TerraForm3D. Based on the freely available OpenGL, GLUT, and GLUI libraries, TerraGUI:

- creates the user interface windows, menus, and other GUI items,
- handles mouse, keyboard, and GUI events, and
- manages, manipulates, and loads visualization data for screen display.

TerraGUI allows the user visually set parameters for rendering distributable and printable visualization products, such as high-resolution perspective view images or VRML (Virtually Reality Modeling Language) files. This saves the user training time by eliminating the requirement to understand complex mathematical 3-space and lighting parameters, which may require much trial-and-error to master. The user is also saved product creation time by allowing the product to be previewed before rendering the final version, which in some cases may take hours. Because of the real-time interactive features, TerraGUI also serves as a visualization tool by itself.

Class Diagrams

Figure 2-1: TerraGUI Class Diagram

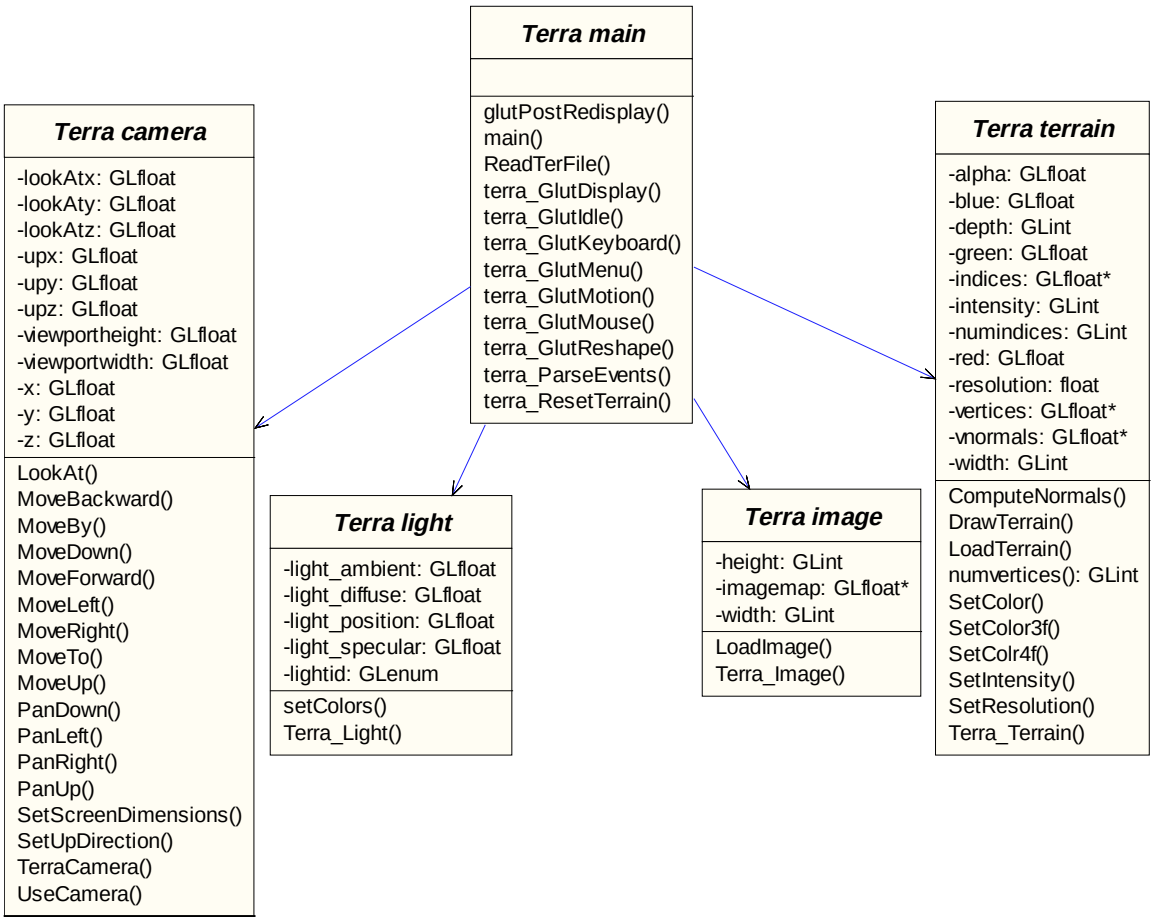
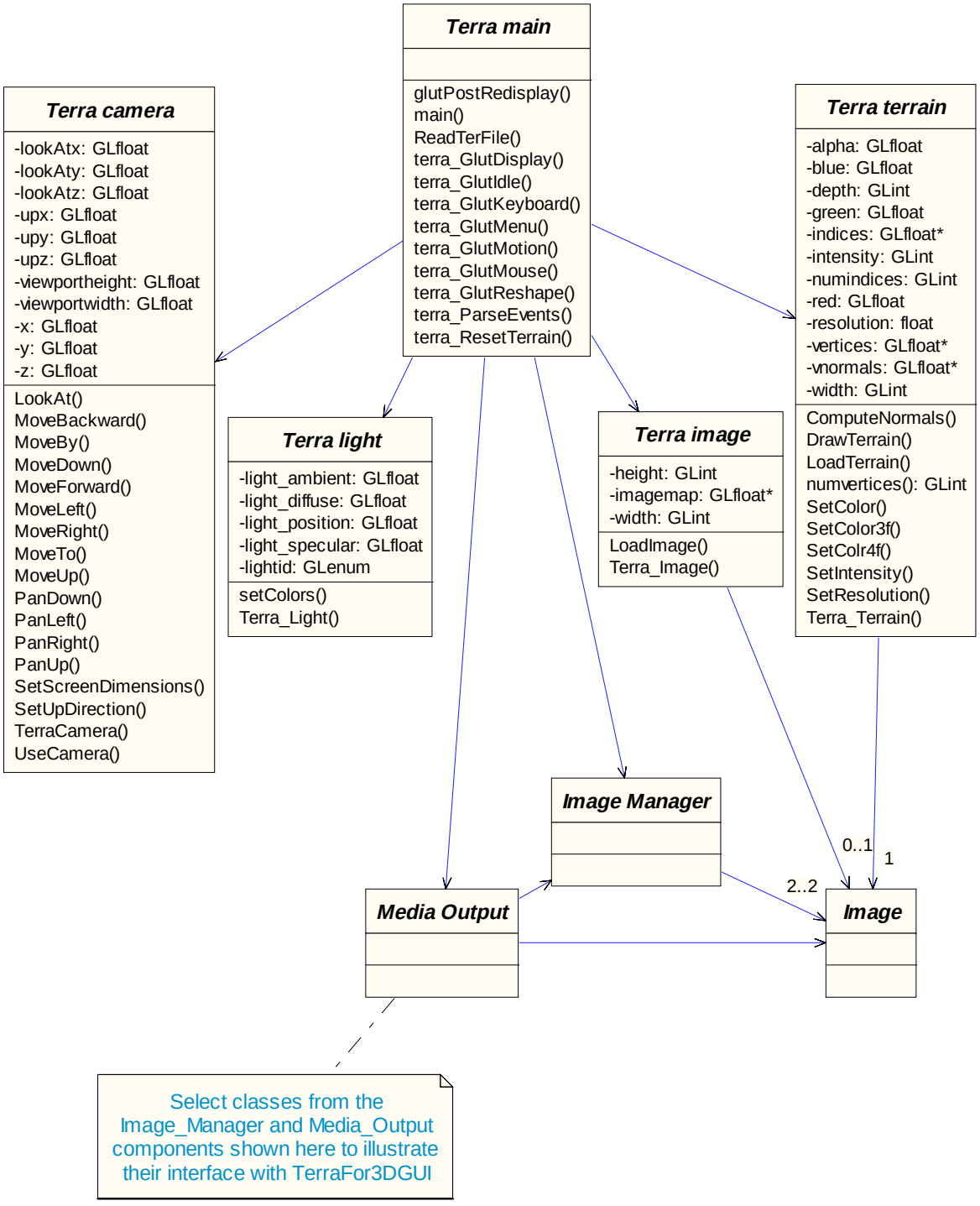


Figure 2-2: TerraGUI Class Diagram – shows relationships to external classes



User Interface

Figure 2-3: Screen shots of TerraGUI user interface

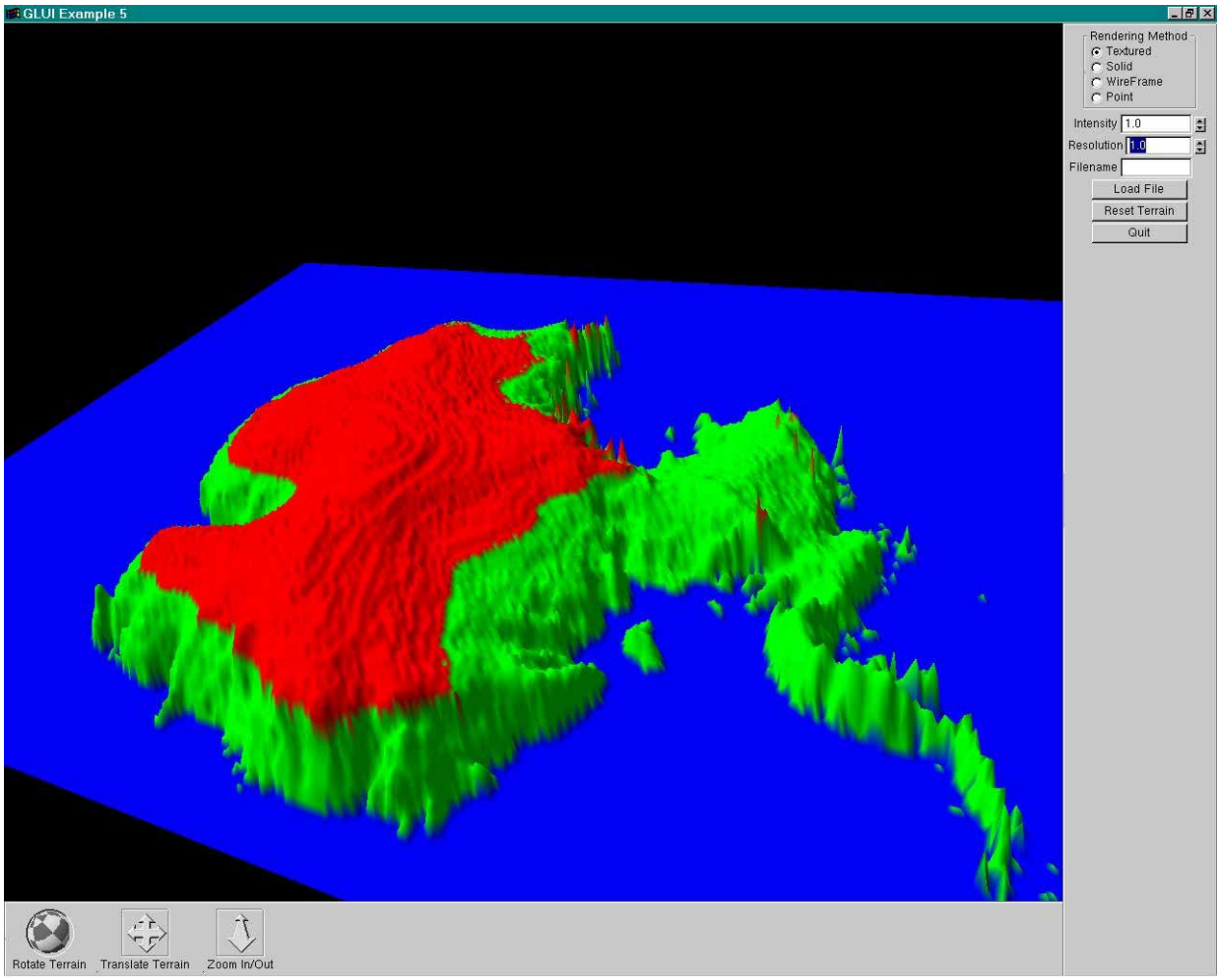
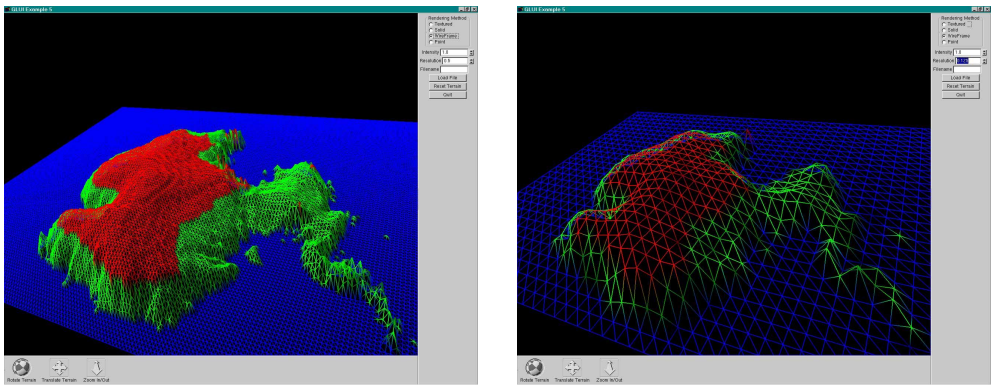


Figure 2-4: High and low resoultion wire-frame mode screen shots



TerraGUI Package Documentation

terra_camera

File	terra_camera.h
Purpose	This file describes the camera that will be used in the terraform3D project.
Author	Craig Post
Last Modified	1.12.2000

FUNCTIONS

Terra_Camera();
Camera defaults to a point at the origin, and is located 5 away on z axis. The up direction is 0,1,0 (y axis is up).

Terra_Camera(GLfloat xloc, GLfloat yloc, GLfloat zloc, GLfloat lax, GLfloat lay, GLfloat laz, GLfloat ux, GLfloat uy, GLfloat uz);
This constructor allows the user to init the camera to their cares.

void UseCamera();
This function causes the camera to be used. It sets up all of the necessary projection matrices.

void LookAt(GLfloat lax, GLfloat lay, GLfloat laz);
Point the camera at this location.

void SetUpDirection(GLfloat ux, GLfloat uy, GLfloat uz);
Set the up direction.

void SetScreenDimensions(GLfloat w, GLfloat h);
Set the screen width and height.

void MoveTo(GLfloat xloc, GLfloat yloc, GLfloat zloc);
Move the camera to this location.

void MoveBy(GLfloat xoff, GLfloat yoff, GLfloat zoff);
Move the camera by this amount.

void MoveForward(GLfloat dist);
Move the camera forward by distance.

void MoveBackward(GLfloat dist);
Move the camera backward by distance.

void MoveLeft(GLfloat dist);
Move the camera left by distance.

void MoveRight(GLfloat dist);
Move the camera right by distance.

void MoveUp(GLfloat dist);
Move the camera up by distance.

void MoveDown(GLfloat dist);
Move the camera down by distance.

void PanLeft(GLfloat angle);
Pan the camera left by angle.

void PanRight(GLfloat angle);
Pan the camera right by angle.

void PanUp(GLfloat angle);
Pan the camera up by angle.

void PanDown(GLfloat angle);
Pan the camera down by angle.

terra_image

File terra_image.h
Purpose This file describes the 2D images that will be used in the terraform3D project. (i.e. textures, icons, bitmaps.
Author Craig Post
Last Modified 1.12.2000

FUNCTIONS

Terra_Image();
This is the default constructor.

Terra_Image(GLint w, GLint h, GLfloat * pixels);**
This constructor takes a preconstructed image and loads it in.

void LoadImage(GLint w, GLint h, GLfloat * pixels);**
Load in an image.

terra_light

file terra_light.h

Purpose This file describes the light data structure. I envision only directional lights being used, but maybe others will be added for effect. This class is intended to be a wrapper of sorts for GL_LIGHTx's. Right now I'm only implementing directional lights because that is all the terrain modeler really calls for.

Author Craig Post

Last Modified 1.12.2000

FUNCTIONS

Terra_Light();
The default constructor.

Terra_Light(GLenum lid, GLfloat * l_amb, GLfloat * l_dif, GLfloat * l_spec, GLfloat * l_pos);
This constructor loads in the color values.

void SetColors(GLfloat * l_amb, GLfloat * l_dif, GLfloat * l_spec, GLfloat * l_pos);
Loads in the colors.

terra_terrain

file terra_terrain.h

Purpose This file describes the actual terrain that will be displayed. It is primarily a data structure but it will need stuff to redefine the resolution on the fly.

Author Craig Post

Last Modified 1.12.2000

FUNCTIONS

Terra_Terrain();
The default constructor

void LoadTerrain(GLint numinds, GLint numverts, GLfloat ** verts, GLint ** ind, GLint wdth, GLint dpth);
Load the terrain using a series of arrays.

void SetColor(GLfloat r, GLfloat g, GLfloat b);
Set the color of the terrain.

void SetColor3f(GLfloat r, GLfloat g, GLfloat b);

void SetColor4f(GLfloat r, GLfloat g, GLfloat b, GLfloat a);

void DrawTerrain();
Draw the terrain. NOTE: all modifications to the terrain are done by the GLUI and so don't need to translate the vertices here.

void ComputeNormals();
Compute the normals of the terrain for lighting.

void SetIntensity(GLfloat i) ;
This function sets the intensity of an object.

void SetResolution(GLfloat res);
Set the resolution of the terrain and reinit the indice array.

Breakdown of Tasks and Data Management

Tasks

Class	***Terra Camera***	***Terra Light***	***Terra Image***	***Terra Terrain***	***Terra Main***
Purpose	Handle information and events for manipulating cameras (ie. viewpoints)	Handle lighting of terrain	Handle image for overlay of terrain	Handle terrain vector data	Create GUI objects on-screen and handle input events
Tasks	Choose camera to use as adtive view Move camera in various directions Pan camera Set camera to look in a direction Hold screen dimensions	Create lighting sources of various types Set colors of lighting sources	Load image	Draw terrain Compute normals Load terrain data Set colorization of terrain Set detail level of terrain	Refresh screen with latest view of terrain Load files Handle keyboard events Handle mouse events Handle screen object events

Data Encapsulation

Class	***Terra Camera***	***Terra Light***	***Terra Image***	***Terra Terrain***	***Terra Main***
Purpose	Handle information and events for manipulating cameras (ie. viewpoints)	Handle lighting of terrain	Handle image for overlay of terrain	Handle terrain vector data	Create GUI objects on-screen and handle input events
Data	Look at x, y, or z Up x, y, or z Position x, y, z screen dimensions	Ambient value Diffuse value Specular value Position I.D. of light source	Dimensions Image map	Color values Indices & number of indices Intensity Detail level Nornals Dimensions	Data related to mouse, keyboard, and other event handlers User data, such as files in use, or data input from GUI elements

Section 3: Image Manager Design

Contents

Figures

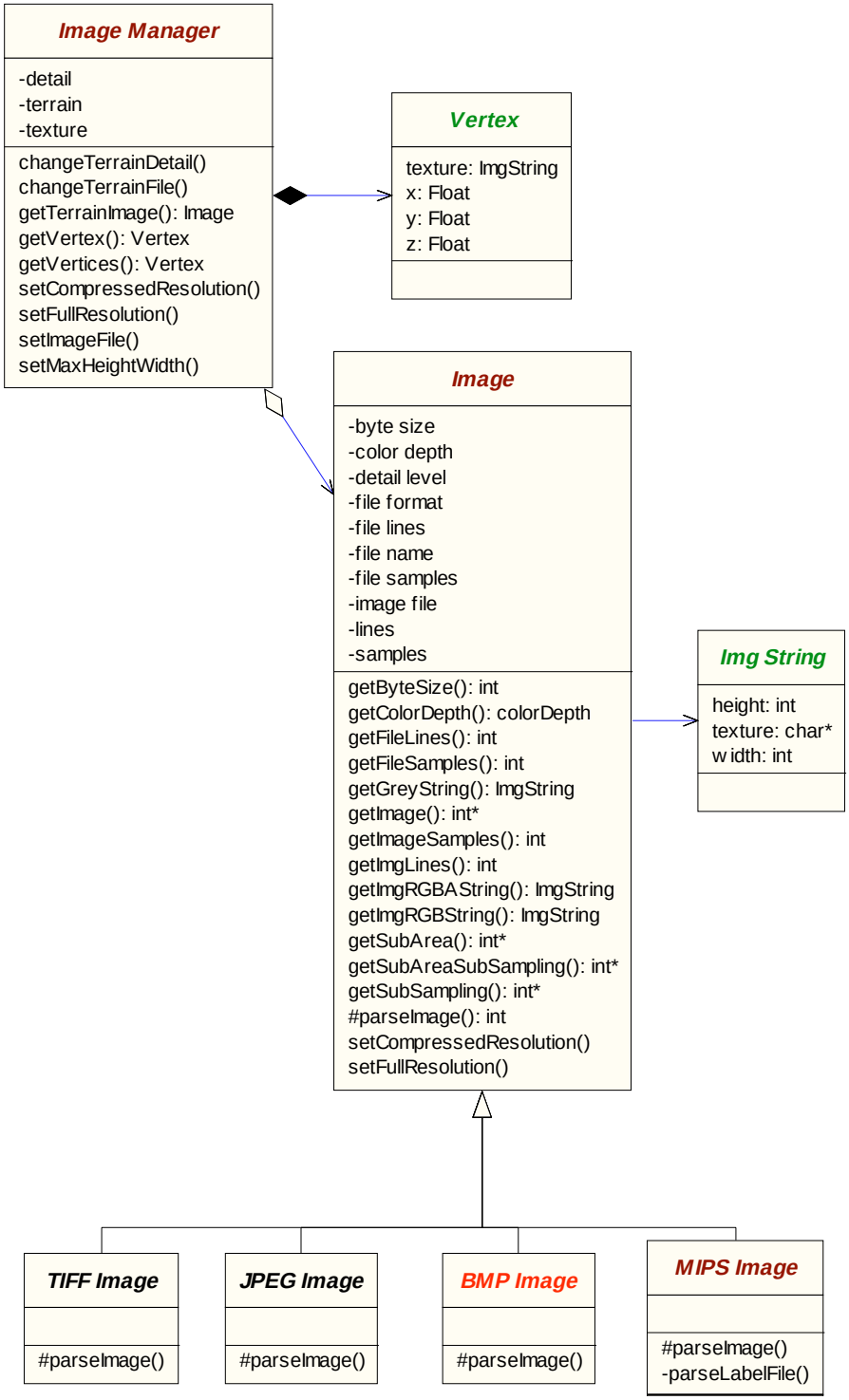
QuickView

Image Manager serves as an interface between image files and other components by reading and returning image data in a generic format that does not require other components to know the format or compression algorithms for a specific image file type.

The package keeps track of a terrain file and an image overlay file, and can return data in a variety of array types. External components to request specific sections of a file at varying levels of detail. Memory management is facilitated by reading from the file every time a request is made, instead of holding the entire image in memory. This type of dynamic image management is vital in a modeling environment where files may be on the order of gigabytes in size.

Class Diagram

Figure 3-1: Image Manager Class Diagram



ImageManager Package Documentation

IMAGE MANAGER

file imageManager.h

class ImageManager

purpose manage a terrain object and texture image data files

GLOSSARY

image a two dimensional array of evenly spaced *pixels*

pixel the smallest element of an image represented by a numerical value which can be displayed on a computer screen as a color. Color images have three values at each pixel – a red value, a green value, and a blue value, which the monitor uses to display a single color on the screen. Grey-scale images have only a single value for each pixel. Typically, the greater a value, the brighter it is.

image file a file which stores image information and data (pixel values). There are a variety of standard image file types. Uncompressed files store data sequentially in a specific order, and can be read directly from the file to a data structure. Compressed files use an algorithm to reduce the number of actual data values stored in the file, and must use a decompression algorithm to extract the data from the file. Both types have some form of header information describing the image's attributes, such as height, width, and byte size.

Image an object used to handle various image file types generically. It holds basic image file information, such as height and width, and reads image data from the file and stores it in generic data structures.

terrain A two dimensional array of evenly spaced values representing terrain elevation (technically, a rasterized digital elevation model). Because the data is stored like image data, it can be displayed on a monitor.

texture An image which can be "draped" over a terrain, such as a corresponding satellite image of the terrain area.

FUNCTIONS

ImageManager ()
Default constructor.

ImageManager (char * terrainFile, char * textureFile, char * textureFile2, char * textureFile3)

Usage:
ImageManager (char * terrainFile) ;
Open just a terrain file
ImageManager (char * terrainFile, char * textureFile) ;
Open a terrain file and a texture image which is contained in a single file (e.g. color JPEG or TIFF)
ImageManager (char * terrainFile, char * textureFile, char * textureFile2, char * textureFile3) ;
Open a terrain file and a texture image which is contained in three different files by band (e.g. MIPS images):
- textureFile = red band file name
- textureFile2 = green band file name
- textureFile3 = blue band file name

Constructor which takes file names. A terrain file is mandatory. Texture overlay file(s) are optional.

ACCESSING IMAGE OBJECTS & DATA

Image * getTerrainImage ()
Returns a pointer to the terrain Image object

Image * getTerrainImage ()
Returns a pointer to the terrain Image object

Vertex * getVertex (int row = 0, int col = 0)
Returns a single vertex at a particular point in the terrain

Vertex ** getVertices ()
Returns a complete set of vertices for the terrain

Other data access functions are contained in the Image class

MANAGING DETAIL LEVELS

In the each of the following descriptions, **<image>** is either Terrain or Texture.

Functions:

```
void setTerrainFullResolution ()
void setTerrainCompressedResolution ()
void setTerrainResolutionCompression (float compression)
void changeTerrainDetail (int height, int width)

void setTexutureFullResolution ()
void setTexutureCompressedResolution ()
void setTexutureResolutionCompression (float compression)
void changeTextureDetail (int height, int width)
```

void set<image>FullResolution ()
All image data will be read out of file

void set<image>CompressedResolution ()
Image data will be read out of file at reduced detail level based on value set using set<image>ResolutionCompression**.** Default compression is 20% (0.2)

void set<image>ResolutionCompression (float compression)
Set the detail level for reading data from files. Default value is 0.2 (20%), which will cause data to be read from file at one-fifth of the full detail level available.

void change<image>Detail (int height, int width)
Sets maximum height and width of image array

CHANGING FILES

void changeTerrainFile (char * filename)
Open a different file for reading terrain data from

void changeTextureFile (char * filename)
Open a different file for reading texture data from

VERTEX

file imageManager.h

class Vertex

purpose Struct-like container for holding terrain data as vertex information and information, a texture array index, and where in the texture the vertex maps to

float x x coordinate = image row ; increments as a portion from 0.0 to 1.0

float y y coordinate (for terrain, is elevation) ; pixel value in file

float z z coordinate = image column ; increments as a portion from 0.0 to 1.0

float u associated row location in texture; increments as a portion from 0.0 to 1.0

float v associated column location in texture; increments as a portion from 0.0 to 1.0

NOTES

Data held by a Vertex is public. Access it like a struct, e.g.

```
Vertex vert () ;
vert.x = 1.0 ;
vert.y = 2.0 ;
```

IMGSTRING

file image.h

class imgString

purpose Struct-like container for holding an image as a character string

PUBLIC DATA MEMBERS

int width the width of the image

int height the height of the image

char * texture the image data

NOTES

Calls to the Image class will return this struct with grey-scale, color, or color with alpha channel image data. Member variables are public – access them like a struct.

IMAGE

file image.h, image.cpp

class Image

purpose Image objects are instantiated by ImageManager, which determines the file type and instantiates the correct child class. Image eads image data from file – child classes hold parseImage algorithm for specific file types – and makes the data available in a variety of structures. Internally, the data is held in a three dimensional integer array, which can be accessed by calling getImage()

GLOSSARY

image a two dimensional array of evenly spaced *pixels*

pixel the smallest element of an image represented by a numerical value which can be displayed on a computer screen as a color. Color images have three values at each pixel – a red value, a green value, and a blue value, which the monitor uses to display a single color on the screen. Grey-scale images have only a single value for each pixel. Typically, the greater a value, the brighter it is.

lines a horizontal section of an image a single pixel high. The *number of lines* in an image is its height or number of rows

samples the data elements (also, *pixel*) in a line of an image. The *number of samples* refers to the number of samples in a line of an image, in other words, the width of the image.

compression and subsampling setting a compression level for the image in the context of the Image and ImageManager classes refers to setting a lower detail level for reading image data. If the compression is less than 1.0, the image data will be *subsampled,* meaning that the data will be scaled down as it is read from file. For example, compression = .5 will scale data to one half the width and one half the height of the original image. This is accomplished by only reading every other sample in every other line of the file. This method is known as *nearest neighbor resampling.*

image file a file which stores image information and data (pixel values). There are a variety of standard image file types. Uncompressed files store data sequentially in a specific order, and can be read directly from the file to a data structure. Compressed files use an algorithm to reduce the number of actual data values stored in the file, and must use a decompression algorithm to extract the data from the file. Both types have some form of header information describing the image's attributes, such as height, width, and byte size.

image data array Member of Image object: three dimensional integer array containing reqeusted data read from file. This array is empty until a request for data is made using getSubArea, getSubSampling, getSubAreaSubSampling, or get*String functions.

NOTES

- Use ImageManager to get access to Image objects.
- **Memory Management:** Image manager and Image do ***NOT*** manage memory allocation and de-allocation automatically. Call **`Image::cleanUp()`** to delete image data array before requesting new data. If using **`int *** Image::getImage()`** to access data directly, **delete the int*** pointer** when finished with the data array.

FUNCTIONS

void setFullResolution ()
All image data will be read out of file

void setResolutionCompression (float compression)
Set the detail level for reading data from files. Default value is 0.2 (20%), which will cause data to be read from file at one-fifth of the full detail level available.

int getImgLines ()
get the count of the number of lines in the image data array (the height of the array)

int getImgSamples ()
get the count of the number of samples in the image data array (the width of the array)

int getFileLines ()
get the count of the number of lines in the image file (the height of the image)

int getFileSamples ()
get the count of the number of samples in the image file (the width of the image)

int getByteSize ()
get the byte size of each element in the image file. The image array stores the data as elements, but the file itself may store the data as one or two bytes, signed or unsigned

colorDepth getColorDepth ()
returns an enumerated type:
colorError = 0
grey = 1
color = 3 (red, green, blue)
alpha = 4 (red, green, blue, alpha)

int * getImage ()**
returns a three-dimensional array of integers containing the image data read from file using getSubArea, getSubSampling, getSubAreaSubSampling, or get*String functions. The data is stored [line][sample][color].

- The lowest value for each index is 0 (zero)
- The highest is [getImgLines() - 1][getImgSamples - 1][colorDepth - 1]

int getSubArea (int startLine, int startSample, int height, int width)
get a full resolution section of the image starting at *startLine* and *startSample*, and of *height* lines and *width* samples. Get the entire image by requesting 0, 0 as the *startLine*, *startSample*, and *getFileLines()* and *getFileSamples()* for the *height* and *width*

int * getSubSampling (float compression)**
get entire image at specified compressed detail level

int * getSubSampling (int height, int width)**
get entire image scaled to fit in a height by width array

int * getSubAreaSubSamp (int startLine, int startSample, int height, int width, float compression)**
get a section of the image starting at *startLine* and *startSample*, ending at *startLine + height* and *startSample + width*, scaled to specified compression. Note *height* and *width* are the full resolution values

int * getSubAreaSubSamp (int startLine, int startSample, int height, int width, int destHeight, int destWidth)**
get a section of the image starting at *startLine* and *startSample*, ending at *startLine + height* and *startSample + width*, scaled to fit the destination array of *destHeight* by *destWidth*

void cleanUp()
deletes image data array.

IMGSTRING RETURN FUNCTIONS

There are six functions, where ***<color depth>*** is **Grey** or **RGB** for greyscale or color. If a greyscale image is requested using an RGB function, each pixel value will be repeated to maintain the RGB reading order.

Functions:

imgString * getImg*RGB*String ()
imgString * getImg*RGB*String (int startLine, int startSample, int height, int width, float compression)
imgString * getImg*RGB*String (int startLine, int startSample, int height, int width, int destHeight, int destWidth)
imgString * getImg*Grey*String ()
imgString * getImg*Grey*String (int startLine, int startSample, int height, int width, float compression)
imgString * getImg*Grey*String (int startLine, int startSample, int height, int width, int destHeight, int destWidth)

imgString * getImg<color depth>String ()
gets full image as an imgString. Values are stored in the struct in a single dimensional character string, in (red, green, blue) order

imgString * getImg<color depth>String (int startLine, int startSample, int height, int width, float compression)
gets section of an image as an imgString. Values are stored in the struct in a single dimensional character string, in (red, green, blue) order. *compression* has a default value of 1.0 (full resolution).

imgString * getImg<color depth>String (int startLine, int startSample, int height, int width, int destHeight, int destWidth)
gets section of an image as an imgString scaled to fit *destHeight* by *destWidth*. Values are stored in the struct in a single dimensional character string, in (red, green, blue) order.

Breakdown of Tasks and Data Management

Tasks

Class	*Image*	*MIPS Image*	*TIFF, JPEG, etc.*
Purpose	Abstract class for reading requested image data from file	Inherits from Image. Reads MIPS format images.	Inherit from Image. Reads files based on specific image formats
Tasks	Get image attributes Get image data as integer array or character string Read image file (abstract)	Read label file Read image file	Read image file

Data Encapsulation

Class	*Image*	*MIPS Image*	*TIFF, JPEG, etc.*
Purpose	Abstract class for reading requested image data from file	Inherits from Image. Reads MIPS format images. MIPS images are composed of a header file and a separate data file, which may be 8- or 16-bit. Full color images are actually three separate such image files, one for each of red, green, and blue channels.	Inherit from Image. Reads files based on specific image formats
Data	Image lines (rows) Image samples (columns) File name Image data Detail level Color depth Byte size File format	Green file name Blue file name (Image:: File name serves as the file name for the red channel)	(most formats get all data from abstract Image class)

Section 4: Media Output Design

Contents

Figures

QuickView

Media Output manages the classes responsible for rendering output products. Data is retrieved from the Image Manager, and rendering parameters are set by the TerraGUI interface. Media Ouput uses these two types of information to instantiate the class type containing the desired rendering algorithms, and feeds it the appropriate data needed to render the product.

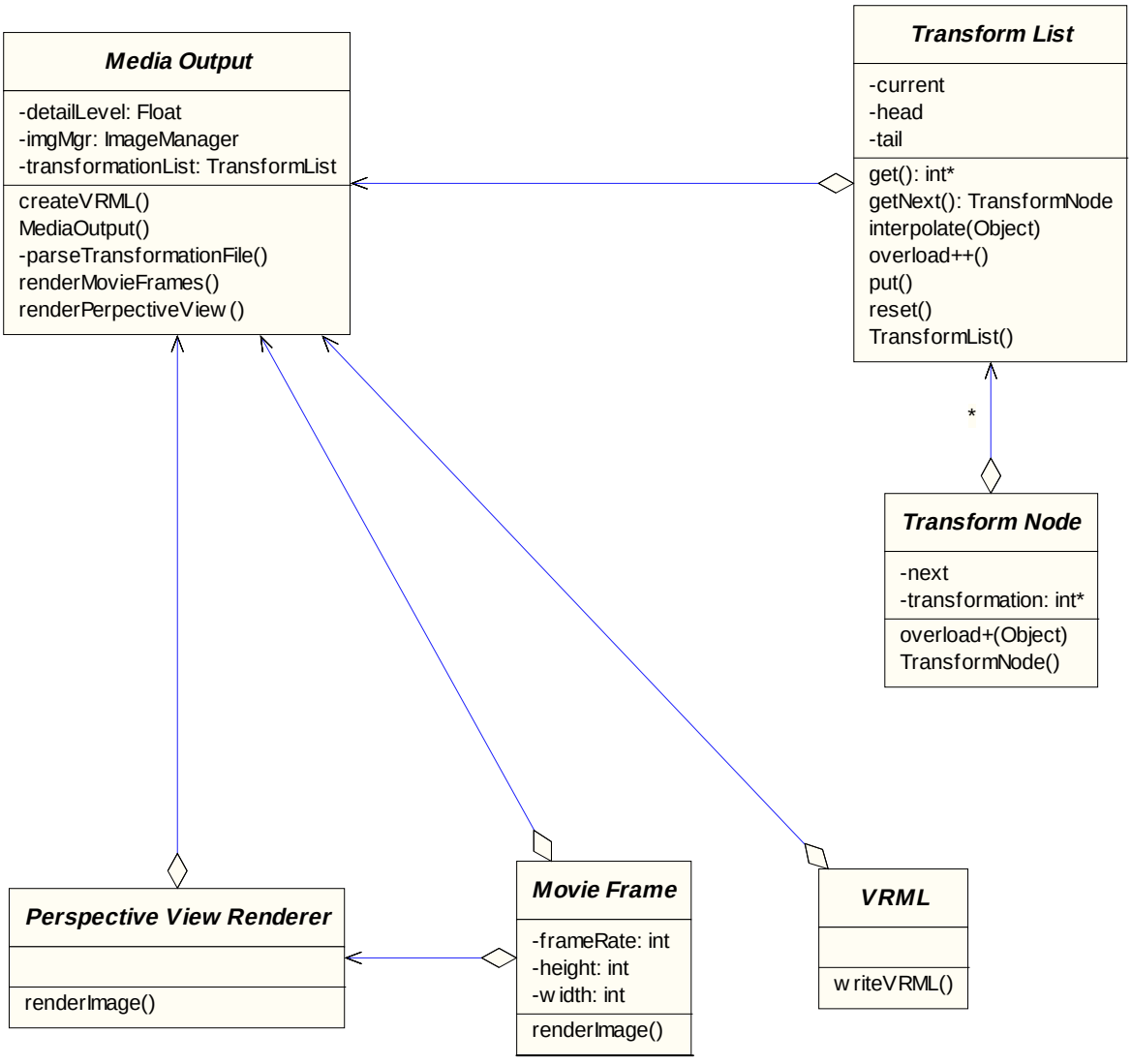
- Perspective View: a high-resolution, two-dimensional rendering of a three-dimensional space as seen from a particular viewpoint, with single-point perspective. The file output is an image.
- Movie Frames: a series of low-resolution Perspective Views that can be assembled using digital movie creation software, such as QuickTime, to create a movie. The image series can depict a fly-by, rotation, or other animation of a terrain.
- VRML: a file type defined by the Virtual Realtiy Modeling Language (VRML) specifications. A VRML file holds low-resolution terrain and image data. It can be displayed using interactive VRML viewer software, which allows the user to interact with a 3D rendered terrain.

High-resolution Perspective Views are suitable for printing on high-quality printers or output to film for photographic printing. Low-resolution Perspective Views are suitable for hardcopy documentation, and in addition to digital movies, and VRML, digital documentation such as web pages and CD-ROM publications. Media Output will be able to be compiled to run on a single

processor workstation or a parallel processing cluster.

Class Diagram

Figure 4-1: Class Diagram of the Media Ouput Package



Media Output

file MediaOutput.h, MediaOutput.cpp

class MediaOutput

purpose manages the classes responsible for rendering output products. Data is retrieved from the Image Manager, and rendering parameters are set by the TerraGUI interface

FUNCTIONS

Derive(...)
A shaded relief is applied to the image. Each computer will do a different part of the image. This is called by passing -function 1 and 10 for right to left, 11 diagonal, or 01 for left to right.

renderPerspectiveView (...)
applies a transformation to a terrain using the PerspectiveView class

renderMovieFrame (...)
creates a series of movie frames using MovieFrames class

createVRML (...)
creates a VRML file using the VRML class

parseTransformationsFile (...)
parses a file containing transformations saved by the user

TransformationList & TransformationNode

file TransformationList.h, TransformationList.cpp

class TransformationList and TransformationNode

purpose holds a series of transformations in a linked list

FUNCTIONS

get, put, next, etc.
standard list functions and data members

interpolate (int num)
creates a series of transformations that interpolate between two given transformations, using overloaded operator +

overloaded operator ++
iterate through list

PerspectiveView, MovieFrames, and VRML

file PerspectiveView.h, PerspectiveView.cpp, MovieFrames.h, MovieFrames.cpp, VRML.h, VRML.cpp

class PerspectiveView, MovieFrames, and VRML

purpose render the given file type

FUNCTIONS

create given file type
each class has a single function called by MediaOutput for rendering the given output product

Breakdown of Tasks and Data Management

Tasks

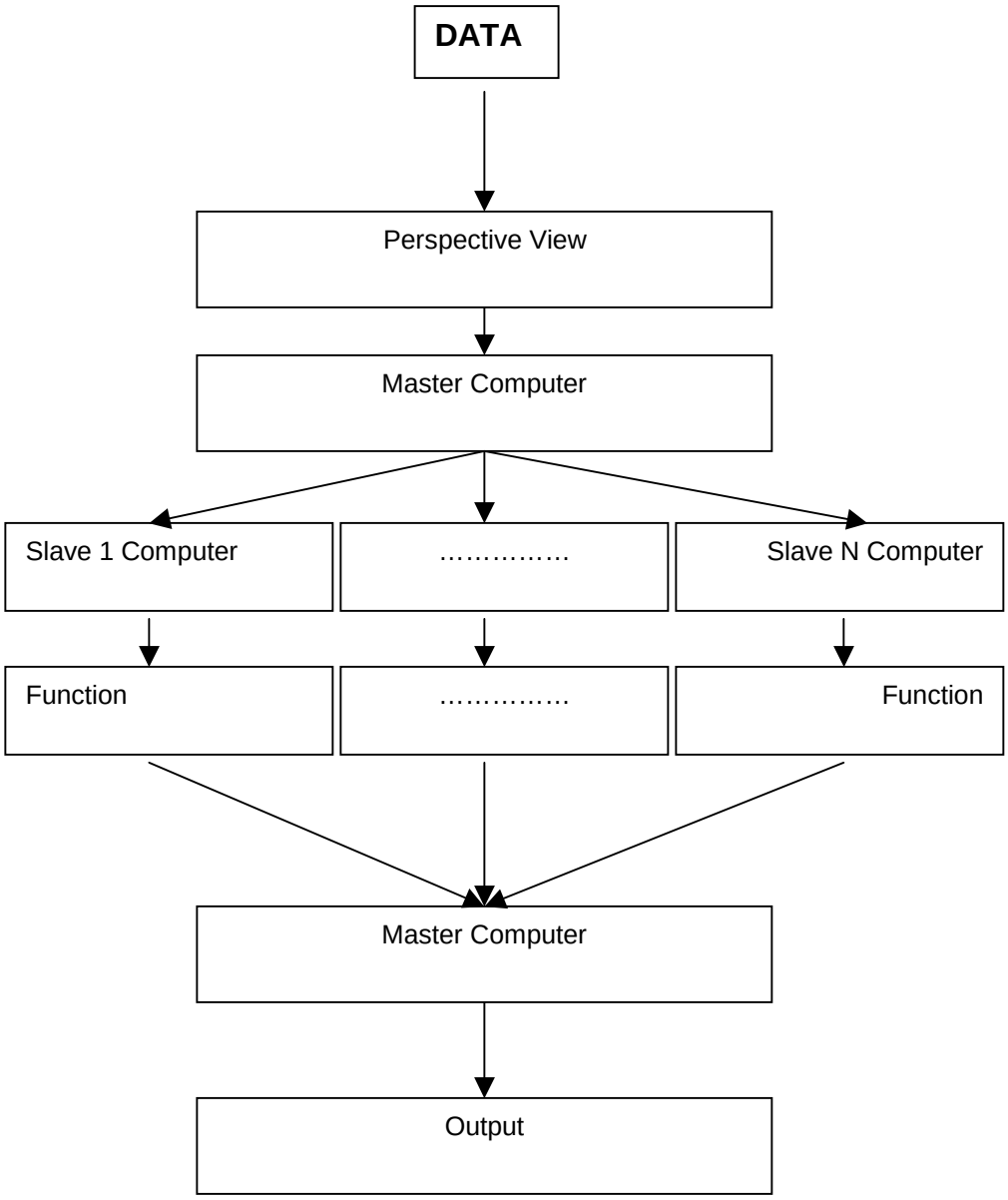
Class	***Media Output***	***Perspective View***	***Movie Frames***	***VRML***
Purpose	Manage data shared collectively by the subclasses Instantiate correct class with appropriate data to create desired output file	Generate 3D perspective view of terrain data	Generate movie frames of animated fly-by of terrain	Generate VRML file of terrain data using transformation matrices to set up preset camera views
Tasks	Collect data Call Perspective View Call Movie Frames Call VRML	Set detail level Apply lighting Transform terrain data Write output image file	Create several low resolution perspective views based on a series of transformation matrices describing the views along a flight path (uses Perspective View to generate output images)	Set detail level Set lighting attributes Set camera view attributes Set other transformation, color, and behavioral attributes Write elevation grid Write overlay image

Data Encapsulation

Class	***Media Output***	***Perspective View***	***Movie Frames***	***VRML***
Purpose	Manage data shared collectively by the subclasses Instantiate correct class with appropriate data to create desired output file	Generate 3D perspective view of terrain data	Generate movie frames of animated fly-by of terrain	Generate VRML file of terrain data using transformation matrices to set up preset camera views
Data	Transformation matrices Output filename Detail level Image Manager (for accessing overlay and terrain data) Color-code (elevation colorizing)	(gets data from Media Output management class)	Frame rate (gets data from Media Output management class)	(gets data from Media Output management class)

Data Flow for Parallelized System

Figure 4-2:Data flow of Media Output on a parallelized cluster of computers



Parallel Function added to MediaOutput

file Parallel_fnc.h

class Parallel_fnc

purpose To find out what kind of work needs to be done in parallel and to separate the work out to all computers

In addition to the serial functions of MediaOutput, the following will added to manage to the parallelizaiton of file rendering

parallel_fnc(nit MaxMegs)
Default constructor. You want to pass the maximum memory that you want used by a computer

runit(int argc, char **argv)
This will get everything. It will assign initiate the MPI calls, get the id to each computer and determine the status. This will then call the function master and slave based on the computers id. The file name and what you are doing will have to be passed in. The file name must be preceded by -fname and the function must be preceded by \function, and any parameters that should also be passed along with it.

> Example of image island having derive done on it from left to right.
> -fname island -function 1 10

master(int argc, char **argv)
This function will only be done by the master computer. It will read in the information about what file is going to be worked on and what is going to be done to it. After this is done it will join the other computers in the slave function.

slave(int argc, char **argv)
The master computer will tell all the other computers what file is going to be worked on and what is going to be done. Each computer will figure out based on its id what part of the image it will be dealing with. TheFunction is then called and the work is done.

TheFunction()
This will decide based upon what is passed to runit() what function will be run, such as Perspective View.

Section 5: Beowulf Cluster Design

Objective:

- **Produce a cost effective solution to image process in a reasonable time.**
  - Use off the shelf computer parts to keep down cost
  - Use free software
- **Measuring objective**
  - Performance - Compare the Beowulf cluster to DEC Alphas (133, 266, and 600 MHz optimized for image processing) doing the same process on an image.
  - Cost - Currently free, all computers are surplus

Hardware:

- **CPU - Intel based hardware, Pentium class**
  - Due to the cost of these products a cluster using this kind of hardware is very cheap.
- **Hard Dive - Small IDE drives**
  - The nodes in the cluster will be using a Network File System to store information on the master computer. This allows very small system disks to be used because only a minimal install will be used. Because swapping is not recommended due to speed considerations in a cluster a swap partition of only 2 times the memory is used. If the cluster proves a viable solution the master computer should be using a LVD SCSI disk so that it can keep a sustained through put.
- **Network card - Netgear FA310TX**
  - This network card is used because of its use of a DEC chipset that gives good performance under a heavy load. Also this card will work at 100 Mbs in full duplex mode.
- **Networking - A 3Com 8 Port Office Switchesign**
  - Using a switch allows each node to get the full bandwidth possible while not causing collisions with traffic from other nodes. This is only possible though if the switch has a big enough back plane to store and send this data for all nodes at the same time. Else the switch becomes the bottleneck, not the collisions of packets. This would be replaced in a larger node system with a high end Cisco switch that could handle more information on the back plane.
- **Hardware Failure - Software**
  - The software bWatch will notify the operator if there is a hardware failure. The program will call the appropriate PI functions to deal with removing the dead node from the work load and picking up for it.

Software:

- **OS - SuSE 6.3**
  - This operating system is running Linux kernel 2.2.13-suse that has the NFS 3 and tulip patch for increased performance with the network file system. The network file system is kernel based also to give more performance.
- **MPI - LAM**
  - LAM (Local Area Multicomputer) is a MPI 1.1 compliant implementation along with some MPI 2 functions. This is free to download and comes with SuSE 6.3. This includes libraries for programming in parallel using C, C++, and FORTRAN. This also provides the calls for allowing the nodes to talk to one another.
- **Monitoring - bWatch**
  - This is a program that comes with SuSE 6.3 that monitors the CPU, and the use of memory in buffers and shared areas.

Security:

- **SSH** - A encrypted protocol that allows a user to log in from anywhere and all packets use 768 bit encryption so that passwords and information can not be easily sniffed of the network. It allows for remote logins, and remote file coping.
- **IP Masquerading** - The master computer runs IP Masquerading so that if the other nodes that are on a fake subnet need to reach the outside world they can. This does not allow for computers on the outside to access them though.

Alternatives:

- **Condor - An alternative to MPI**
  - This program would require that no computer know what another computer was doing at the same time. With some of our algorithms this is not possible.
- **Write our own Server/client sockets**
  - Due to our time constraint this was not feasible.

Section 6: Appendix A

TerraForm3D Coding Standards

1. The .h files will all contain the following information at the top of each file

file	:	File name
class	:	*Class name*
purpose	:	*Purpose of the class*
author	:	*Person who wrote the file*
date	:	*Last date worked on (unless using RCS, which will put in the date.)*
history	:	*What has been done (unless using RCS, which will record changes that have been made)*

2. Every function that is declared in the .h file will have a brief description about what it does and how to use it above it.
3. Comments in the .cpp files will be up to the discretion of the author in determining if they are needed.

Section 7: Appendix B

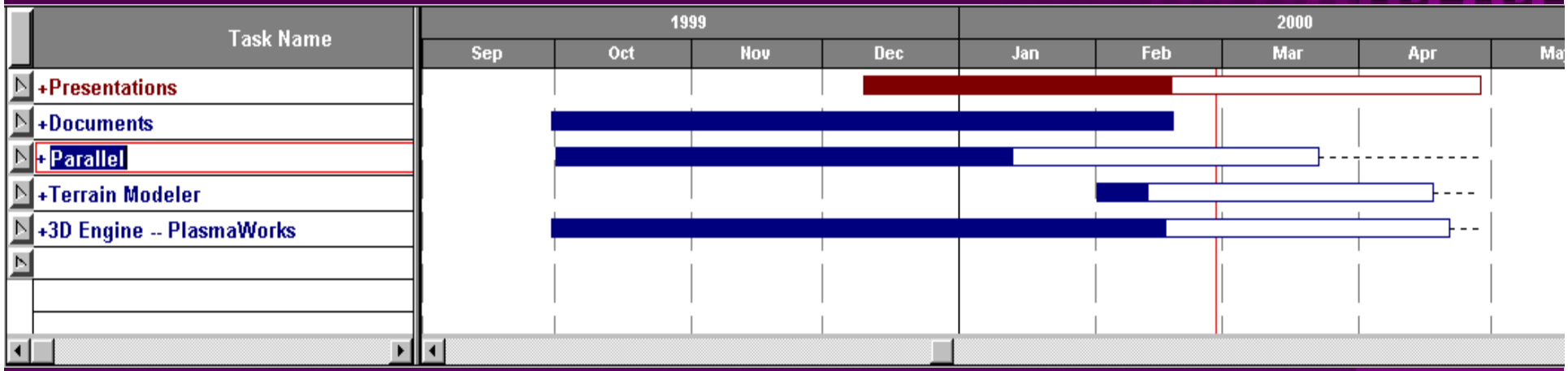
Testing Plan

A checklist of the following tests will be created and maintained to document the testing of the classes, packages, and completed software for delivery of the final product:

1. Each class will be compiled on multiple platforms and tested as it is written to ensure each function works correctly, and be reviewed by another team member for comments, standards, clarity, and correctness
2. All packages shall be tested with the following input files:
    a. Unsigned char MIPS format terrain file
    b. Signed word MIPS format terrain file
    c. Unsigned char MIPS format greyscale overlay image
    d. Unsigned char MIPS format color overlay image
    e. Additional file types as added to the Image class structure
    f. All types with small (25 x 25 pixels) and large (minimum 4K x 4K pixels) file sizes
    g. All terrains with a variety of colorization schemes (4 to 256 colors)
3. ImageManager has been tested on a DEC Alpha running True64, and an IBM-compatible running SuSE Linux, using the applicable tests in point 2
4. MediaOutput, serial and parallel, will be tested on the command line using a test driver and with TerraGUI. The following output will be created using the above input:
    a. Perspective view with a combination of the input files in point 2 above
    b. MovieFrames with given transformations and interpolation of transformations to render a variety of frame rates
    c. VRML (serial only) with and without an overlay texture, and at various levels of detail, camera views, and lighting configurations
5. TerraGUI
    a. has been tested as a stand-alone program with hardcoded, simple input of a small terrain file (200 x 200) on a PC
    b. will be tested with ImageManager using the testing using the input files in point 2 above
    c. will have all events tested by developers for proper functioning, and by outside users for proper functioning and usability

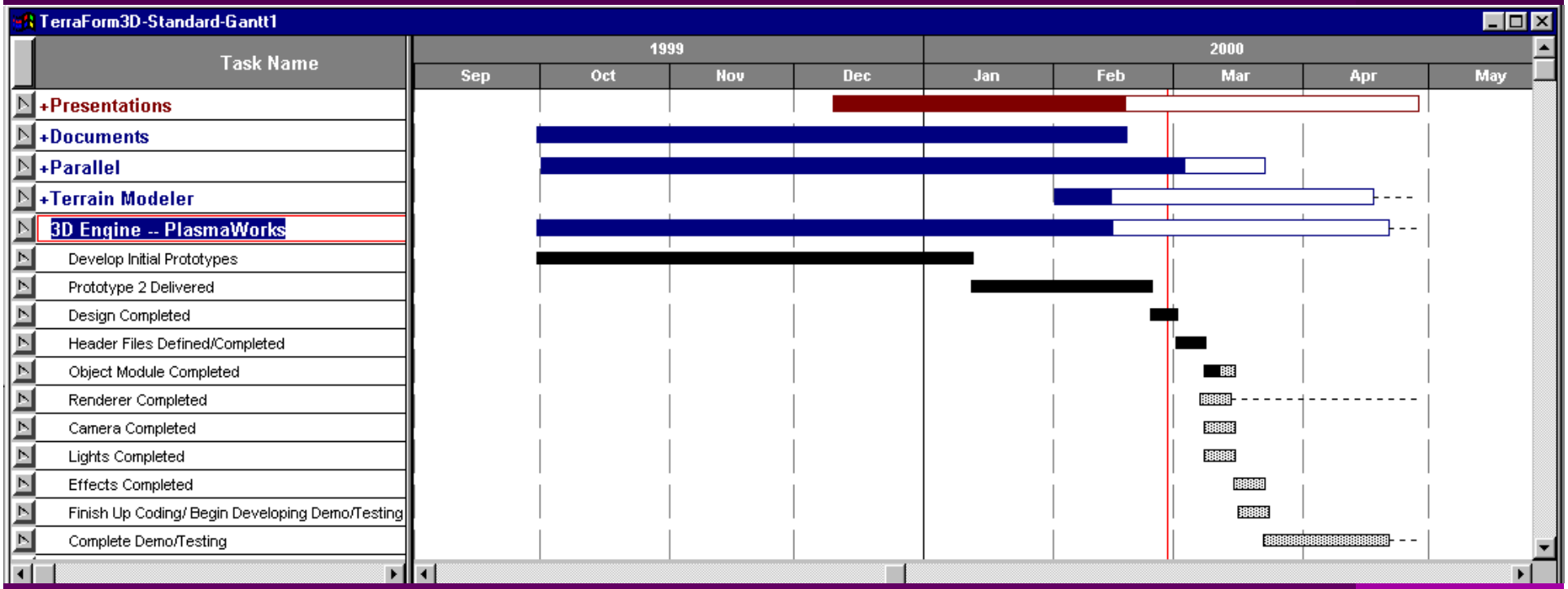
The checklist will contain the platform, operating system, input file names and types, the tester's name, reviewer's name, and date of testing for each component.

Main Task Schedule

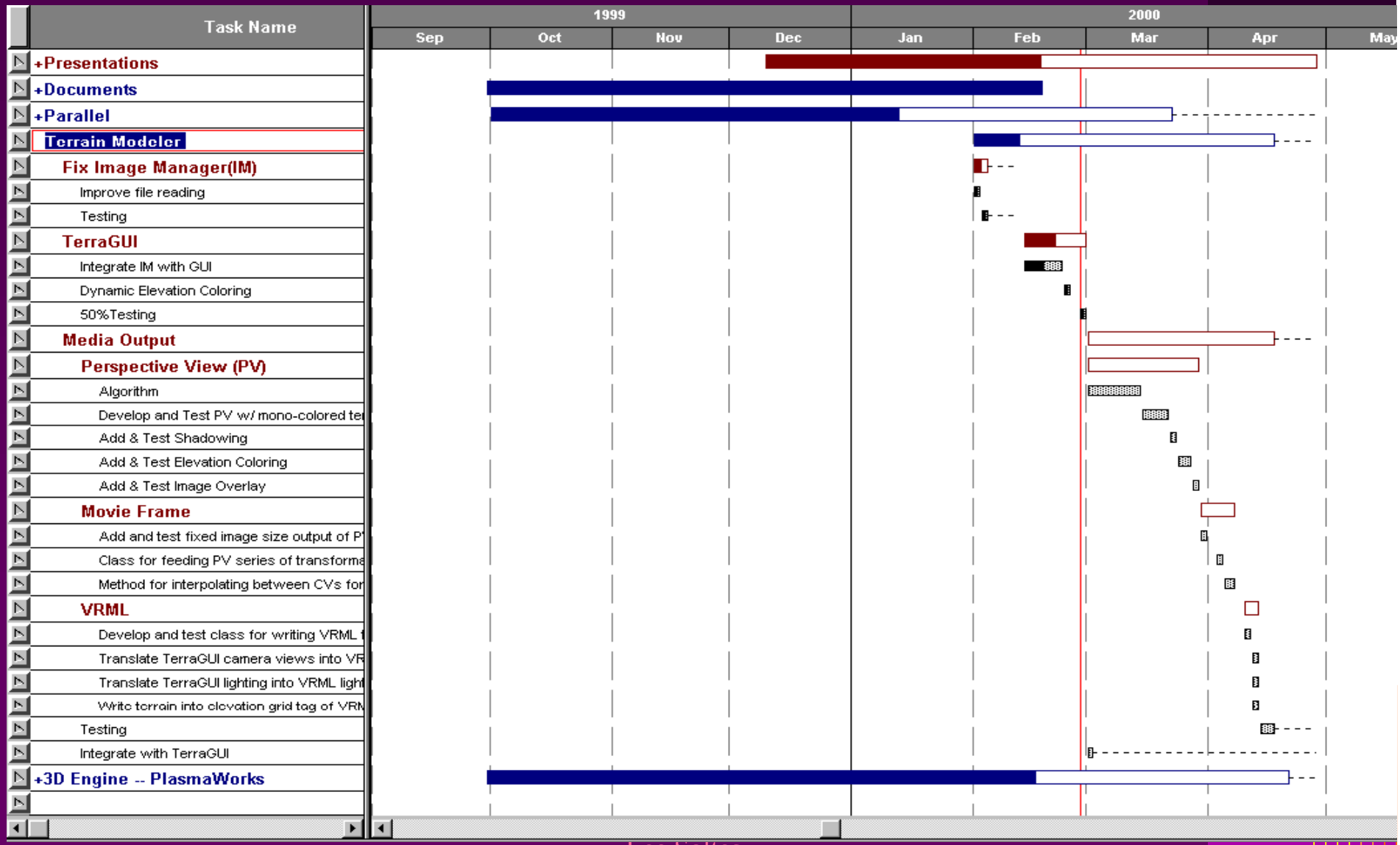


- Simultaneous development of both Engines
- Status: On Schedule
- Finish Code Development by April 1st
- Final Testing: April 2nd – April 20th

Plasma Works



USGS Terrain Modeler



Parallelization