Golf Course Design

Richard Howlett

Computer Science

2003/2004

Summary

The problem the project sets out to solve is how a computer system might be used to assist in the design of a golf course. It will involve aspects of not only golf courses but also terrains in general. It will look into the aspects of multi-resolution terrains and the issues involved. It covers an approach to allowing multi-resolution terrains to be created as well as just displayed. The problem involves issues of functionality and usability. The project aims to satisfy both of these as fully as possible, to not simply allow a user to create a golf course design, but also the assist in this action.

Acknowledgements

Many thanks to Peter Elmy of Elmy Landscapes Ltd. For agreeing to assist in the definition of user requirements and breakdown of the problem and the evaluation of the product produced as a result of this project.

Contents

1) Understanding the problem

To understand the problem it is required to look at the different aspects of the problem, not just the functional requirements of the program but also the method used to produce the program. First and foremost a look at the requirements of the game golf is required. After the main design requirements of the game are realised, the most appropriate methods of implementation will be examined, such as operating system, programming languages and various API's that may assist in the development of the solution.

1.1) Breakdown of requirements

The program needs to allow users to create a golf course on a terrain. This will obviously require looking into any requirements that may exist with the game golf, such as what the program needs to be able to do to allow a course to be specified.

The program needs to support at least simple manipulation of the terrain for creating objects required for golf courses. This will require looking into methods of computer based terrains, such as storage and methods for creating modifiable terrains.

The program must allow users to in some way visualise what the designed course would look like. This requires research into different ways of visualising an environment.

1.2) Computer platform and programming language

Supporting multiple operating systems would obviously be beneficial. This would require using a programming language that supports multiple operating systems, such as java which is platform independent. Since the operating system support will in part be determined by the programming language and available API's there is little to be determined from looking at the advantages of different operating systems.

The main languages available to develop the solution are C/C++ and Java. The main advantage of java would be its platform independence, however there is a lack of available information on 3D application programming with java in comparison to C/C++. Also making graphical based programs on java is more complicated than with C/C++ especially with the number of API's available to assist with these things for the C/C++ languages. As well as the actual merits of the languages there is also personal experience using the languages. Since prior experience has been gained due to graphics modules it would be sensible to take advantage of the existing experience provided. Due to this C/C++ seems to be a very suitable choice, especially coupled with the use of the GLUT API covered in the modules. Also since prior experience is with GLUT an OpenGL API, OpenGL will be the graphics API that will be used. Another advantage of GLUT is its platform independence, GLUT will work with both Windows and UNIX operating systems, without having to change the code.

1.3) Existing solutions and similar projects

1.3.1) Golf course architects

There are many golf course architects now, due to increasing popularity of golf as a sport and also as a result of increasing difficultly in creating courses. Several of these architects offer services where an idea can be presented to them and they will model the designed course using a private program, however this is unavailable to use by people other than the architect so this is simply a service and not a tool to aid design, as most of the interpretation and design will be done by the architect.

1.3.2) 3D studio max

Although not specifically targeted at landscapes, 3D studio max is a very powerful 3D editing tool. It supports many different 3D formats and is designed for any form of 3D modelling and animation. It has a very large customer base in industry and also a large number of companies creating plug-ins for the program to allow easy additions to its initial intentions. The program allows the manipulation of planes into 3D terrains, by various methods such as direct manipulation of the points, or by displacement from a height map. The terrain can be edited to increase or decrease the resolution of the terrain in different areas. The program also allows adding of objects to a modelled terrain, however there is no currant plug-ins to aid golf course design.

The main problem with the program is that it is targeted at professional 3D modellers, is not very user friendly for beginners to computer based modelling and due to the target audience is far too expensive for a large number of people.

1.3.3) Editor42

Editor42 is a terrain editor created by Frederick Taillon for use with a recent computer game called battlefield 1942. The main features of this editor are that it allows the manipulation of a 3d terrain, via real-time flying around the terrain and also by directly editing the height map (see figure 1 (Taillon (2003)). The application also allows texturing of the terrain to add key terrain detail such as roads and fields (see figure 2 (Taillon (2003)), however the program does not allow any placement of 3D objects onto the terrain. The application is simply for manipulating terrains not placing objects.

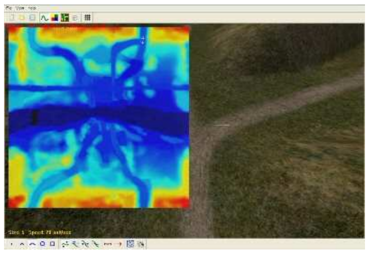


Figure 1: Height map editing.

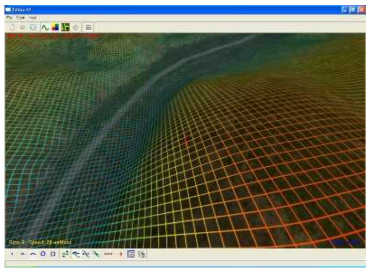


Figure 2: Showing grid and texturing.

There are several implementation problems with this editor is that it uses Microsoft DirectX extensions and therefore only functions on Microsoft operating systems, which is not ideal. A second problem observed is that although it allows very good manipulation of the terrain, the game it is designed for only uses height maps of a certain resolution, this means it does not cater for the requirement to have areas of different resolutions that was identified in previous research. The program is free which presents a large advantage.

1.3.4) Height Map Editor (Hme)

Hme is a 2D based editor which as the name suggests is based around height maps. It is basically a painting program (See figure 3) that provides tools for making areas higher or lower by painting a lighter or darker colour onto the image. The program has tools to set the height to a specific value, raise or lower an area pick an area as the height to set to as well as various other useful commands such as raise and lower entire terrain.

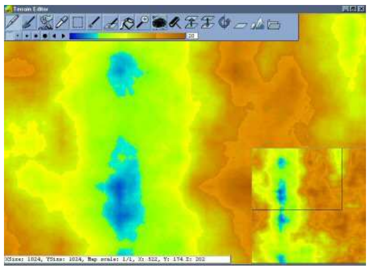


Figure 3, Hme.

The main advantage of this program is that it is very simple to use, all that is required is to draw the height of the different areas, however there is no 3D view in order to examine what the terrain looks like in a real life viewpoint, due to this the program is not very suitable. Also a program like this would then require another mode to actually draw the texture so that the terrain looks how it should.

1.4) Research based on requirement breakdown

1.4.1) Golf rules

The professional golfer's association (2003) states there are five basic skill sets. "The Skill Set System breaks holes down into 5 areas of concentration, based on equipment type and the challenges posed by different parts of the course". Golf therapy retreat and wellness research centre then details that "Alongside or on the fairway are many types of challenges (generally classified as "hazards") designed to cause the golfer to swing frequently: trees, sand bunkers (scooped out areas in which earth has been replaced by sand; balls don't roll in sand very far; a golf course will contain on average 80-100 bunkers), rough (high grass, bushes, trees, flowers, undergrowth), water (lakes, ponds, streams, oceans, marshland), and other (quarries, fountains)."

Other than hazards a course has a tee "box", this is where the golfer starts from, does not have any specific requirements as it can be up to the designer as too terrain conditions. The fairway, this is the area the golfer should aim to stay on and also has no specific requirements although it may need to be slightly more detailed than the rough areas, again this is up to the designer. The last major item is the green; this is where the hole is located and usually has to be far more detailed than any other area of the golf course.

In general the only actual requirement of a golf course is to have a tee "box", a fairway and a green with a hole. Every other aspect of a course is down to the designer. Due to this a tool to aid in golf course design only need to allow the placement of these 4 objects. However it is hazards that make a golf course challenging, so for good golf course design, it must also be possible to place a variety of hazards on the terrain.

There are some other more specific details about golf courses. "Tees establish playability and are prime targets of improvement. Multiple tees are prime targets of improvement. Multiple tees are the norm today due to the wide variety of players. It is not uncommon for tees (four, in many cases) to cover 5,000 to 7,000 sq. Ft. in area on the modern course." The American Society of Golf Course Architects (2001). If pre-made tee boxes are made for the application then this could be used as a rough size to use, this would aid designers that are unsure of the scale of the various parts of the course.
"The shape, size and protecting features of each green should be in direct relation to the approach shot. Although larger than those of earlier eras -- a good average size is 6,500 sq. Ft. -- modern greens should offer variety." The American Society of Golf Course Architects (2003). Again this could be used a basis for any pre-made greens.

The tee being so large is not only to accommodate the large variety of players, but also for purposes of grounds maintenance. By having larger tees and indeed larger greens it allows the various tee off points and the hole for the flag to be moved, this way when certain parts of the course are getting highly worn, the flag etc can be moved to allow the worn areas to be restored

“Cart paths are becoming an increasing necessity. Their proper routing can make the difference between slowing or speeding play. It is imperative that they be incorporated into the overall design of the golf course.” The American Society of Golf Course Architects (2003). Due to this allowing the designer to add paths may also be required, however in Britain it is still common for golfers to walk on foot, with no facilities for golf carts.

There is no specific standard when it comes to flag colours, as different courses are free to use their own colour schemes. Coloured flags can also be used to signify the position of the putting hole relevant to the green, for example, if it is central to the green, tee side, or far side of the green. This is an aid for the player so they know where they need to position their ball ready for putting. As with flags tees also have their own colour system, this is used to signify the tee off position for different skill levels, e.g. beginners, intermediates and professionals. Due to these reasons a variety of flag and tee colours should be available to the designer.

1.4.2) Computer terrains

Obviously any solution will need to store the details of the terrain so some research is required into the possible options available. There are several different methods that are used for storing geometric data; the main ones will be detailed here.

1.4.2.1) The DEM

One common method of storing such data is “DEM” or Digital Elevation Model. This is detailed in The University of British Columbia (1997, 1), “DEM is frequently used to refer to any digital representation of a topographic surface however, most often it is used to refer specifically to a raster or regular grid of spot heights“. Although there are several uses of DEMs the application relevant to this project is its use for determining attributes of terrain, such as elevation at any point, slope and aspect”. DEMs store the height of selected points of the terrain however they use a regular resolution, so areas of high contour density will not have additional points to represent them. Due to this the choice of resolution is very important, if a low resolution is used then slopes with many steps in them will not be stored with this information, instead they will only be recognisable as a constant slope with no steps, see Figures 4-7 for effect of resolution.

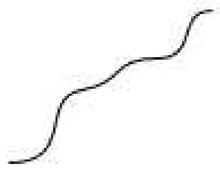


Figure 4, Actual curve

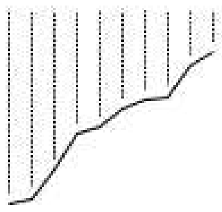


Figure 5, High resolution (10 points)

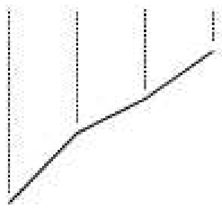


Figure 6, Medium resolution (4 points)

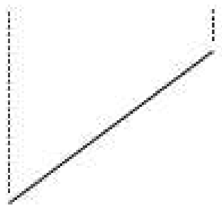


Figure 7, Low resolution (2 points)

The figures clearly show that is the resolution is high then slopes will be stored more accurately, however with lower resolutions slopes may be stored very inaccurately (comparing Figures 4 and 7).

Height maps are simple implementations of DEMs they are used in many modern 3D programs such as many recent computer games. They simply store the height of each sample point and so the resolution of the height map determines how much detail is stored.

This is the simplest method of storing geometric data, however since golf courses have areas of varying detail e.g. the green may need to be far more detailed than the rough, then It will not be suitable. Since either a high resolution would be needed, taking a large amount of space where there may be little change in terrain height. Or a low resolution to save space, meaning only a less detailed green could be stored. Obviously a value in between may be a suitable alternative, however a geometric storage method that could allow differing levels of detail to be stored would be preferable.

1.4.2.2) The TIN Model

An alternative to DEMs is the TIN or Triangulated Irregular Network model. It is “a simple way to build a surface from a set of irregularly spaced points” The University of British Columbia (1997, 2).

“Irregularly spaced sample points can be adapted to the terrain, with more points in areas of rough terrain and fewer in smooth terrain, an irregularly spaced sample is therefore more efficient at representing a surface” The University of British Columbia (1997, 2).

This helps prevent the problem of resolution that is present in DEMs. The TIN model works on the principle of selecting sample points on the terrain that represent the most significant points, then joining them together as triangles. By using triangles the entire surface can be connected into a full surface, without any problems of fitting shapes together. This method of triangles works best with terrains where there are sharp edges present on the slopes, e.g. the slopes are not smooth but instead are flat with sharp corners.

The main problem presented with this method is that triangles must be formed and which triangles are used can at times have a drastic effect on the accuracy of the stored data, this is mainly because the triangles loose a large amount of the curve definition that may exist on the terrain and would also significantly reduce the ability to add fine curves. The other large problem is that points also have a large impact on the stored data too. There are several algorithms for picking points detailed in The University of British Columbia (1997, 2).

This method is more suitable for storing large areas of low height change gradient and also suitable for areas of high gradient, however the smoothness of any such areas of terrain may not be as accurate as a high resolution DEM.

1.4.2.3 The Quadtree

The quadtree is an example of a hierarchical data structure The University of British Columbia (1997, 3) describes the problem that the quadtree tries to solve.

“The amount of information shown on a map varies enormously from area to area, depending on the local variability; it would make sense then to use rasters of different sizes depending on the density of information. Large cells in smooth or unvarying areas, small cells in rugged or rapidly varying areas, unfortunately unequal-sized squares won't fit together ("tile the plane") except under unusual circumstances, one such circumstance is when small squares nest within large ones.”

A quadtree allows detail to be stored in a way so that where areas have a high change in elevation gradient the number of points used to represent the data is increased and where there is a low change in elevation gradient, few points are used to store the data. This provides a more accurate representation of the terrain. When expanded to large scale terrains it presents a large saving in the amount of storage space required.

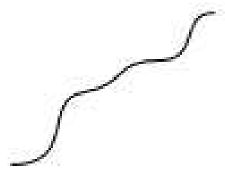


Figure 8, Curve.

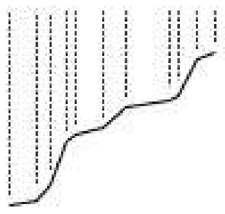


Figure 9, Quadtree.

Since this storage method allows storage of data in multiple resolutions, it would be suitable for storing golf course designs as it is suitable for both areas of little change and areas of high change allowing accurate greens to be stored and large open areas too, without using too much unnecessary space. Unlike the TIN model, quadtrees also maintain detailed curves, so very detailed greens could be created and stored without loosing any slope information, making this the more suitable model for storing a golf course terrain.

Another good advantage of quadtrees is that it is simple to increase the resolution of a particular area. To do this simply requires changing one of the tree leaves into a node which would then contain 4 leaves. This way a simple operation on the tree can convert any given square into 4 smaller ones.

1.4.2.4) The Heightmap

The height map is a single resolution solution. The principle of a height map is to use an image of a certain resolution to define the profile of the terrain. Heightmaps do not allow different points on the terrain to be different distances from each other however; it is a very fixed format. Since the format simply stores a grey scale value to determine the height of the terrain at that area, it is based on a scale principle; a value of 255 (white) is given for the highest points and 0 (black) for the lowest. Any values in between simply specify the height of the point relevant to the highest and lowest point. The heightmap simply defines heights and the resolution of the map defines how many points there are with modifiable heights. How far apart the sample points are is determined by the program.

1.4.3) Visualising terrains

There is one large problem associated with the visualisation of large scale terrains. This is the problem of performance. There are several methods available that can help reduce the problem of performance. Firstly back face culling can be used so that the application only renders the faces that the viewport can actually see; any faces that do not face the direction of the camera are automatically excluded from the rendering of the image, reducing the work load to the graphics card.

A simple one of these to implement is the use of frustum culling. This process allows only the vertices and faces that are in the viewport to be rendered. In a large terrain this could have a very large impact on performance as it significantly reduces what needs to be rendered. This is relatively simple since it involves checking the location of vertices when rendering, the object is checked against the planes that make up the left, right top and bottom clipping planes as well as using the values of the perspective projection that determines the near and far clipping planes. If an objects lies within the frustum it is rendered, if not it is skipped. Because only the required vertices are rendered there is a relative maximum amount of data that will ever be displayed at the same time, once this maximum has been met, further increases to the size of the terrain will have little to no effect on the performance of the program, and this is obviously a very good thing.

There are also several complex algorithms that are aimed at simplifying the data to be parsed to the rendering device. Lindstrom et al (1995) discusses one such method, “Perspective projection causes distant polygons to appear smaller on the screen than polygons close to the viewer. At some distance, the vertices that make up a polygon are all going to render into the same pixel on the screen.” The algorithm proposed by Lindstrom et al (1995) “determines the correct distance at which a smaller set of polygons may be used to approximate the terrain surface. Currently the reduced data set is created by decimation of every other grid elevation point, resulting in a factor of four reduction in the number of polygons that must be rendered.” The algorithm is then run further as the viewport’s distance gets greater, further reducing the number of polygons to render.

1.5) Possible approaches to the problem

There are several different options for a 2D based solution, the first of these is to have several views so that the terrain can be specified using isometric views, this may be rather complicated to use and would probably be rather difficult to visualise 3 dimensionally.

The other main option is to have a height map based editor similar to the Hme program discussed in section 1.3.4. This would require one mode to edit the height of the terrain and then another mode to paint the actual real visual textures to specify the areas of the green, fairway etc. Again like with the previous option it may be very difficult to visualise the terrain in a 3 dimensional manner.

A 3-dimensional application already starts with the advantage that visualising the design in a real life perspective is already far simpler due to the fact that the terrain will be displayed in 3D anyway. A 3D mode would also allow any number of angles.

Conclusions

The research has shown that there are not very many real requirements for a program to assist in the design of golf courses except the ability to place trees, flags and tees and to also specify areas that are part of the green, fairway, water, sand and rough. Most of the requirements will be determined by how these operations can be assisted.

2) Design methodology and project plan

2.1) Methodology

The methodology that will be used will be an approach based on a prototyping life cycle. This model involves several stages, with one section being an iterative process. Bennet et al (1999) states that there are several main stages behind this process. Initially an analysis needs to be done, this is to determine what needs to be achieved by the program and to allow the prototyping process to remain structured, "Embarking upon a prototyping exercise without some initial analysis is likely to result in an ill-focused and unstructured activity producing poorly designed software." The next stage is to define the prototype objectives. This stage requires defining what the prototype needs to accomplish, this is important to ensure that prototype iterations provide useful improvements. The following 3 stages are part of an iterative loop, firstly specifying the prototype; this requires specifying what each prototype will do. The next stage of the loop is to construct the prototype; this obviously involves creating the prototype. The last stage, evaluate prototype as well as recommend changes, is to examine the suitability of the prototype and recommend what changes are required in order to better fulfil the prototype objectives. Bennet et al (1999) states, "The purpose of the prototype is to test or explore some aspect of the proposed system." This is key to the process that shall be followed. If changes are required the 3-stage section is iterated, returning to the specification of the prototypes and continuing to evaluation again. This is done until the evaluation deems the product to be acceptable in meeting the prototype objectives. Bennet et al (1999) suggests several advantages of the prototyping approach. "early demonstration of system functionality help identify any misunderstandings between developer and client;" also it is simple to identify what requirements have not been met, as well as any difficulties with the interface that may be present.

There are several problems with the prototyping approach, ones of these is that "the prototype may divert attention from functional to solely interface issues;" and also that the client may not understand the further requirements to make a complete product and may perceive the prototype as a finished product. To help reduce these problems, early evaluations will be carried out by more technical evaluators, whilst the later evaluations will be carried out by the industry evaluator. This will be discussed later in the evaluation section of the report.

To a certain extent the project will also share some methodology with an incremental development approach, in that the program will initially start small with extra functionality being added as the project progresses. This approach selects the features of the requirements that will most fulfil the user requirements initially, in this case the minimum requirements of the project, then as time allows extra requirements will be met, these being the various extensions possible as well as meeting the minimum requirements to a greater degree.

There are several deliverables that the process will create, firstly will be a list of functional requirements this will be created from the analysis of the minimum requirements and the project objectives. Following this will be to define the objectives of the prototypes to be created. Next the iterative loop will be entered, in this stage; the prototypes will be defined, implemented then evaluated and repeated. Once the initial prototypes have been completed the final program will be started, by final in the context of this product it is meant the combination of the prototyping outcomes to produce a more complete program, this will follow the same bases as the prototyping procedure as well as the aspects of incremental development discussed. The final stage will be the evaluation of the final product identifying the final state of the product.

The following is a summarised list of the deliverables.

- List of requirements the program needs to meet.
- Details of the prototypes, as well as what they must accomplish.
- Completed prototypes that achieve the prototype objectives set for them.
- A final product that meets at least all the minimum requirements of the project and preferably more.
- Evaluation of the final product at whatever position it is at the end of the project schedule.

2.2) Project schedule

The following is a schedule that is based on the deliverables identified in the previous section. The schedule contains some more detailed breakdown of the deliverables, as to what and when intermediate stages need to be completed in order to create the deliverable.

- Product requirements
    - Conduct preliminary research. (13/10/2003→ 12/12/2003)
    - Conduct preliminary interview with industry fellow to assist determination of user requirements. (10/11/2003)
    - Analyse research to determine product requirements. (12/11/2003 → 19/11/2003)
- Create prototype objectives
    - Investigate the requirements of the prototypes. (19/11/2003 → 24/11/2003)
    - Define what each prototype must accomplish. (19/11/2003 → 24/11/2003)

- Complete prototype programs
    - Complete first prototype. (24/11/2003 → 12/12/2003)
    - Complete second prototype. (28/11/2003 → 05/01/2004)
- Complete a product for final evaluation.
    - Combine features of first and second prototypes. (05/01/2004 → 26/01/2003)
    - Use incremental development methodology to develop a product that meets minimum requirements, and then expand to extra functionality. (26/11/2003 → 19/03/2004)
    - Refine program based on final evaluations (19/03/2004 → 19/04/2004)
- Evaluate the product
    - Mid project evaluation (Technical).
    - Easter evaluation 1 (Technical and interface).(24/03/2004)
    - Easter evaluation 2 (Technical and interface).(07/04/2004)
    - Finish write-up of evaluation (19/04/2004)
- Complete report
    - Complete write-up of complete report (13/10/2003 → 28/04/2004)

Due to several reasons the schedule was delayed at several points. One delay was the difficulty of arranging a suitable meeting time for the evaluations during the Easter period with the external evaluator this resulted in both evaluations being one week behind schedule (31/03/2004 and 14/04/2004 respectively). This also delayed the completion of the write-up by one week. Periods of illness also delayed the schedule most notably the completion of the program ready for evaluation was set back by 2 weeks, this also had the effect of having to use a less than optimally completed program for the first evaluation and obviously a less developed final product than originally planned could be evaluated. The final report write-up was also delayed for this reason by 2 weeks.

3) Development of the solution

In this section the design methodology will be followed to proceed through the design and implementation of the solution.

3.1) Initial Analysis

The following are the objectives of the project.

- Create a test application that allows the manipulation of a terrain to be suitable for a golf course and allows the placement/arrangement of objects onto a terrain. This means that it will be possible to have any items required for a golf course on the terrain, in whichever way this may be implemented.

- Create a prototype that allows creation/modification of golf based objects such as the tee box (where the golfer starts), the fairway (the area the player should aim to stay in) and the green (where the hole is located). By creation/modification this means that it will be possible to have golf objects of the required size and shape on the terrain, in whichever way this may be implemented.
- Create the final application that combines the two prototype applications into one functioning application. This final application will allow the manipulation of a terrain to allow the placement of golf based objects to be placed onto the terrain and modified in some way.

These requirements can be broken down into a series of design requirements. Firstly there are several functional requirements that can be extracted from the above information:

- It must be possible to move points on the terrain.
- The detail of the terrain must be changeable.
- It must be possible to specify features on the terrain, e.g. the green.
- It must be possible to place objects on the terrain.
- It must be possible to remove objects.
- It would be preferable if the user could have a first person perspective of the terrain.
- It should be easy for the user to view specific areas of the terrain.
- It should be easy for the user to make out small differences in terrain elevation.

There are also several usability requirements. Most of this is based on the concept that the tool should aid the designer and not hinder them.

- Appropriate functions must be easy to find.
- The user must be able to make a course without having significant problems.
- The commands should behave in a manner that is consistent.
- The commands should behave in a manner that is expected from the name of the command.
- It should be obvious how to use a command, e.g. what to select on the screen.
- The procedures for the operations to the terrain should be simple.
- The operations should perform the required effect.
- The use of commands should be easy to learn (Low program learning curve).
- It should be easy to learn how to use the operations to achieve the required result.
- The terrain should be easy to modify.

- It should be simple to add features (e.g. green, sand pits) to the terrain.

The list detailed should contain any requirements for the program. These features will mostly be covered by the prototyping approach, since most of them are required for the program to meet the minimum requirements, however since there are extensions as well as minimum requirements, there are some features that will be mostly covered in the incremental development approach of the project. These are determined from the minimum requirements and the extensions of the project.

The minimum requirements are:

- A program that allows users to create a golf course on a terrain.
- Allow simple manipulation of terrain for creating objects required for golf courses.
- Allow user to in some way visualise what the designed course would look like.

The possible extensions are:

- Logging of geometric changes to facilitate golf course costing. This would help calculate the cost of any physical landscape changes required. As well as possible budget mode that would allow user to only make changes up to a specified value.
- Automatic landscape modification to level terrain to the shape of specific objects. This would allow for example a green to be made in advance as a template and then placed on the terrain, which would then mould the terrain to fit the template green.
- 3D flyby mode to automatically show the terrain created.
- Implement a par (the standard number of strokes to complete the course) calculator to state the par of the designed golf course.
- 3D golf game that allows the user to try out the golf course design, by playing through the golf course.

The following is a list of the requirements that would be preferable to add to the program, these mainly represent features that would enhance the program beyond simply meeting the lowest level requirements of the program; many simply have relevance to making the program easier to use.

- It should be possible to move areas of points as well as single ones.
- The detail of large areas should be changeable.
- There should be good control over the level of detail.
- It may be required for the points to be moveable in all 3 dimensions.
- It would be preferable to do able to do this for large areas.

- The user should be able to position features anywhere on the terrain.
- It is preferable if objects can be placed on large areas.
- It is preferable if the exact position of terrain objects can be specified.
- The user may require control over objects placed, e.g. colour and size.
- It may be required to be able to place a selection of different types of trees, plants and bridges etc.
- It would be preferable if large areas of objects can be removed.

3.2) Prototype Objectives

The program development will be split into several stages; the 2 initial stages will be to create 2 prototype programs that will concentrate on the two main technical aspects of the project. The first prototype will cover the aspects of creating a terrain where the heights of the vertices can be modified. This will also cover the placement of objects onto the terrain as well; this prototype will concentrate on the operations that need to be performed on the vertices. The second prototype will concentrate on the operations that affect faces; this is the ability to sub-divide squares and then the ability to define the areas that belong to the main course components, e.g. green, fairway etc.

3.2.1) Prototype 1

This prototype concentrates on vertices, as a result it is first important to decide how the vertices should be stored and what necessary methods will be required with their use. The terrain will need to have some form of storing all the points on the terrain, to allow the operations to work easily and make it simple to find adjacent vertices, a fixed sized grid will be used in this prototype. Since this is the first prototype it requires first looking into the process of object selecting. The first task is to review object selection methods and create a working selection system. Simply the process requires catching the mouse click events, then running several functions, first the mode is set to select instead of render, then as usual projection mode is selected, then gluPickMatrix is used to define the area of the screen to take samples from, this defines a window to look at, then the perspective is set, the mode then needs to be set to model view, and the scene rendered. Once this has been done the render mode needs to be set back to render and the hits assigned to a pointer, this allows the results to be looked though, lastly if there are hits, then a function is called to process the hits and determine what was selected on the screen. In order to differentiate vertices from each other glPushName and glPopName are used to give each vertex a unique name. This program gives each vertex 2 names, firstly its row number, then its column number. This allows the program to determine which location to look at in the terrain array. This is an 11 x 11 x 3 array, where there are 11 rows, 11 columns and then 3 values, the x, y and z co-ordinates of the point. The next objective for this prototype is to allow the program to place objects on the terrain and remove them. This will be done using the same selection method as above, but instead will require an entry in a placed objects array to be changed to indicate the location of a tree.

3.2.2 Prototype 2

The prototype objective for this prototype is to explore the possibility of a multi-resolution terrain. This consists of being able to change the level of detail in the required areas of the terrain. This will require the creation of a new class to hold the information required. There are several things that will be required for this class, the class will be discussed in more detail later in section 3.3.2.3, but briefly it will need to be able to store 4 sub-trees in case the tree is a node, and also 4 sets of vertices, to specify the 4 faces of the tree if the tree is a leaf. The second objective is to allow the program to change the texture of different areas on the terrain; this will require the tree class also storing the texture for each of the 4 sub-trees so that it knows what texture to use when rendering the faces.

3.2.3) Final program

The objectives of this stage are to fulfil all the functional requirements that relate to the minimum requirements of the project.

The initial stage in designing the final program is to look at the structure with which to represent the data on the screen. The terrain needs to have a suitable number of points on it to manipulate in order to achieve the appropriate level of control over the shape of the terrain. It also requires the functionality explored through the development of the prototypes.

A class will be required to store the information on the vertices in the terrain that is more detailed than the array system used in the prototype. Each vertex will need to store the x, y and z co-ordinates of the point, as well as this, the class that represents the vertices will need to support addition and division so that averages can be taken, this is not required for this prototype, however will be required for the second prototype. This class will effectively be like a standard 3D point class. Further details will be given later in section 3.3.2.2.

3.3) The Solution

As mentioned the creation of the program has been initially split into three main tasks to accomplish, firstly the creation of the prototype to allow the manipulation of the vertices on a fixed resolution terrain. This will also include the ability to place objects on the terrain and remove them. The second task is to create a prototype to allow a single face to be modified so that the resolution of the area can be increased. Finally the last stage in creating the program is to integrate the two prototypes to create the final program that allows the manipulation of the terrain as well as manipulation of the level of detail on the terrain. The final version will also include the ability to change textures to define key areas on the terrain. This section will concentrate on documenting the implementation of the prototypes and the main program.

3.3.1) Prototypes

This will document the results of the prototypes, showing what they accomplished.

3.3.1.1) Height manipulation / Object placement

This prototype program allows modification of terrain heights and loading/saving of modified terrain. It allows placement of trees and loading/saving of object placement data. Figures 10 and 11 show example of lowering the height of a 9x9 area on the terrain (this was done several times for easily noticeable effect. Currently allows manipulation of a single point, a 3x3, or a 5x5 matrix. For matrix manipulation it automatically moves points to try and maintain a curved surface, by moving the points around the selected point less moving away from the selected point. All the commands are available from a menu brought up by clicking the right mouse button (figure x).

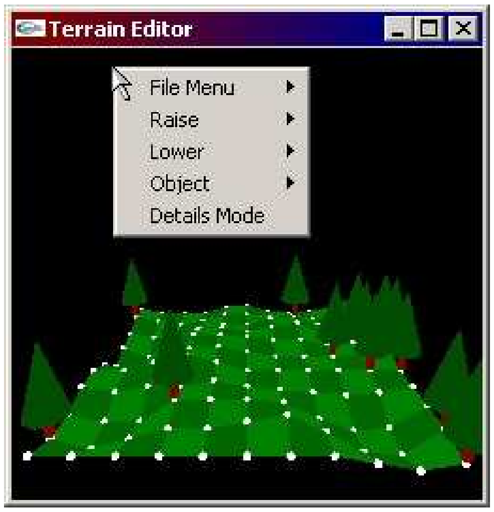


Figure 10, Before, showing the menu.

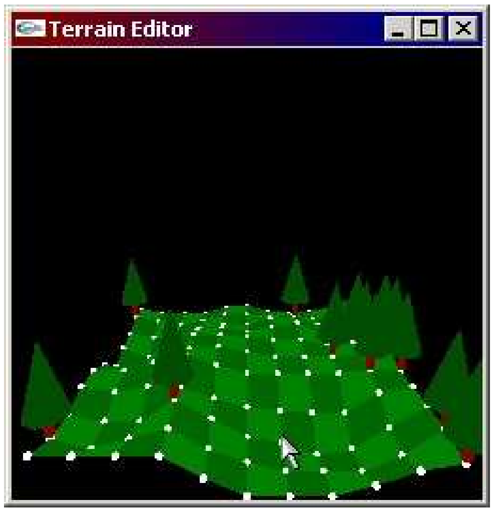


Figure 11: Terrain after using 9x9 lower tool.

The prototype stores all the vertices in an 11 x 11 x 3 element array, the first dimension is the row number, the second is the column number and the third is the x, y and z co-ordinates. When a point is selected it raises the selected point by a distance of 3 units, if the action is raise area, then it looks at the surrounding points, of a distance of one vertex and moves these points by 2 units, finally if raise large area is selected, all the vertices of a distance of 2 points from the selected one are raised by 1 unit. This is to provide a smooth curve. The prototype also has an 11 x 11 x 1 array to store the tree data. Basically in this prototype, the value is set to 1 for whichever vertices on the terrain have a tree one them e.g. if the 3rd tree on the bottom row has a tree, then placedObjects[0][2][0] is set to 1 to indicate the presence of a tree at that position. This is very wasteful since if only one tree is present on the entire terrain, it still needs to store the same amount of data as if all vertices had trees on them. A vector based approach would be far more suitable, where an entry can be added which details which vertices have trees on them. The program also has no support for differing sized terrains. The terrain can be saved, this is done by simply writing the positions of each point to a file. Effectively this is a height map based approach, so the points are always in the same position on the x, z co-ordinate plane. Loading just writes these positions back into the terrain array.

3.3.1.2) Terrain resolution adjustment

This prototype program allows increasing the resolution of areas of the terrain. This is for refining the terrain for making more detailed greens. See figures 12 and 13 for example of terrain being made more detailed. Since this prototype is looking into the sub-division of a terrain it does not allow moving vertices, this will be left till the creation of the final product, where the features of this and the previous prototype shall be joined.

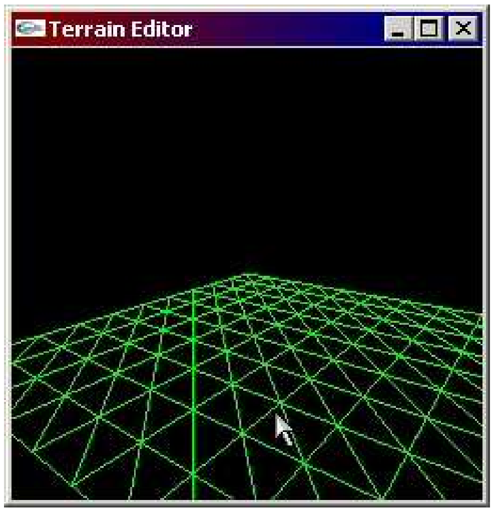


Figure 12, Plain Terrain.

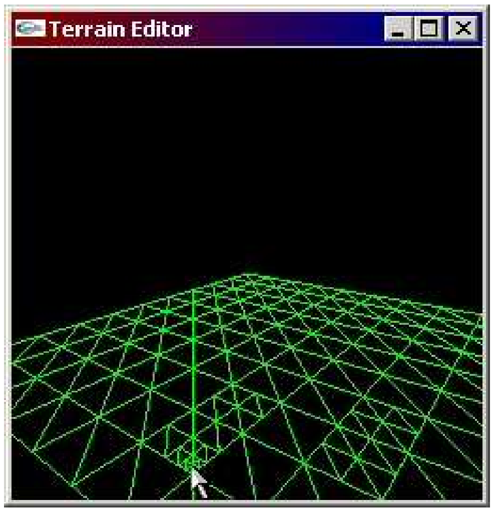


Figure 13, Terrain with higher resolution areas.

This initial version only deals with creating sub-divided faces in the correct area and does not cater for the aspect of sharing vertices. This version of the program automatically creates all the vertices required for the sub-divisions, regardless of whether there is a neighbouring sub-divided face that has a vertex in the same position. The second version of the prototype concentrates on just one face of the terrain. The main aim of this version is to allow the sub-trees to share vertices with each other if required. Figures 14 and 15 show how adjacent sub-trees share a vertex once they are both sub-divided.

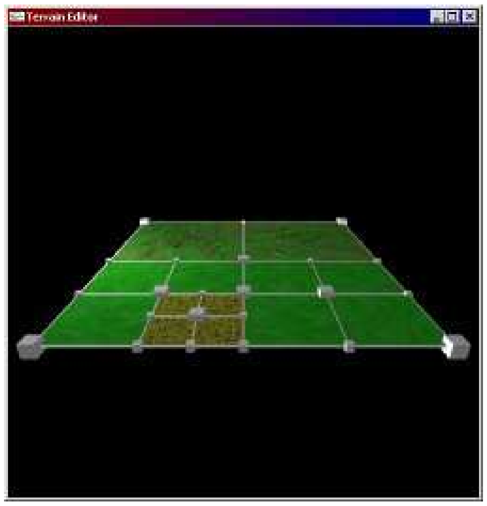


Figure 14.

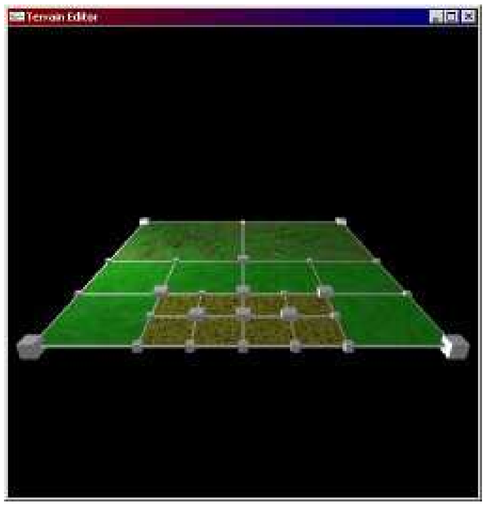


Figure 15.

Figure 15 shows a large square in the position that previously had a small square in figure 14. This is because of several reasons, these will be discussed later in 3.3.2.3 but it shows that sub-dividing the second tree has caused the program to find the existing vertex and use that instead of creating another vertex in the same position.

3.3.2) The final program

The final program being rather large and complex will be described in several sections.

3.3.2.1) The terrain

The terrain has 3 different groups of data that are stored in order to provide all the required data for creating the surface, firstly is a vector array that contains all the vertices used in the terrain. The vertices are simply an x, y and z co-ordinate as well as a Boolean flag to tell the program if this point can be manually moved or not. This is identical to the initially designed class for the vertices except the addition of the moveable flag. The reasons for this change will be discussed in section 3.3.2.2 as well as a complete overview of the class. The second group of data is the information on the faces in the terrain. This is another vector array containing objects of a purpose built struct class. This class contains all the required information for defining one square on the terrain or one face, which consists of 4 triangles. The class contains the index value of the 4 corner points of the face, so if the face uses the first 4 vertices in the vertices vector then it will store 0, 1, 2 and 3, (points defined in an anticlockwise direction when looking from the front of the surface) instead of storing the actual position, this is done so that several faces can all use the same vertex, without having 4 copies of the vertex stored in different places. The face struct also contains the currant face mode, this is important for knowing if the area has been sub-divided or not. Also the texture id for each of the 4 triangles that make up the face are also stored and finally the face index of the faces immediately up, down, left and right from the currant face, the use of this will be discussed later in section 3.3.2.3. A struct was used for this face information instead of writing a full class because the data did not require any functions to manipulate the data. These two groups of data alone are enough to specify the initial terrain that the user has to work with. Since together they store the details of where the points in the terrain are, which points to connect together to create each face, what texture to assign to each of the triangles and which are the neighbouring faces. The last data group is required for making the terrain more detailed, discussed later in section 3.3.2.3. This is the Tree class

The terrain is a grid of faces. The surface can either be made up of 4 triangles (Polygon mode), or it can be made up by a tree (Tree mode). The reason for using a tree is simple, when starting with a fresh terrain, all areas of the landscape are simply drawn as triangles using the vertices that make up that face on the grid. If more detail is required then the mode of the face can be changed from a polygon to a tree, this allows increasing the level of detail in the area of that face. This is the basis of displaying the terrain and allowing detailed areas to be made without increasing the detail on the entire terrain, it allows unmodified areas to remain course whilst heavily worked on areas such as greens can be made detailed.

3.3.2.2) The TreeVector

This is basically the class that represents the vertices on the terrain. It consists of the 4 items of data already discussed as well as several functions to manipulate the data. The class has 3 constructors, allowing a plain TreeVector to be created or allowing the specification of the co-ordinates, or also the moveable flag. It has a function to allow the easy retrieval of the moveable status of the point used in functions discussed in section 3.3.2.3. As well as this it includes a distance function to calculate the distance from one TreeVector to another one and finally it has overloaded +, -, * and / operators to allow commands, again these are used by certain functions described in section 3.3.2.3. There is nothing of note in the implementation since it shares most of the same principles as other 3D vector classes around.

3.3.2.3) The Tree class

The trees are designed so that when subdividing a square, it produces 4 sub-trees within the space of the original. The Tree class contains the details of the four sub-trees that replace the currant one, the texture type of each of the 4 sub-trees, the currant mode of each of the sub-trees and the index values of the 4 vertices that make up each of the 4 subdivided positions. A tree can either be in leaf mode or node mode. If the tree is a leaf, the program uses the 4 vertex indexes for the tree to draw the two triangles that compose the square, if a tree is in node mode, then the tree contains 4 leaves, these are all trees that are in leaf mode. So essentially a Tree contains 4 sub-trees, and the mode can be flipped for each of the 4 sub-trees to determine the level of detail in that area.

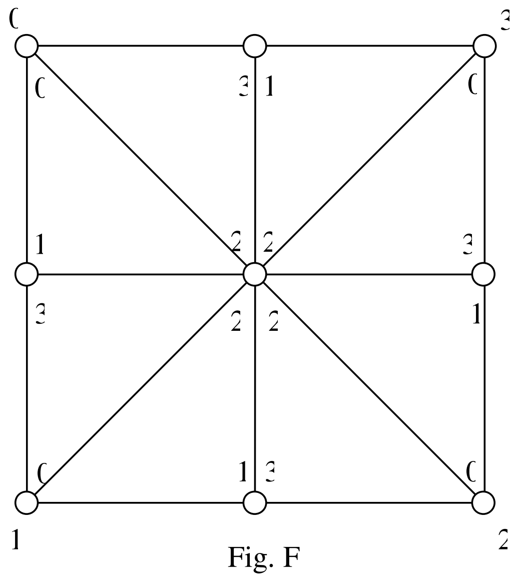


Fig. F

The structure is designed in such a way that for the sub-trees contained within the tree, the first vertex is always the corner point of the main tree, they are then determined in an anti-clockwise direction from the first point. This allows the polygons to always have the ridge from the outer corners to the centre point and also to allow certain calculations to be made more simply. The subdivision of a square creates 4 smaller squares instead of splitting 4 triangles in half as discussed by Lindstrom et al (1996) in a paper on “Real-time, continuous level of detail rendering of height fields”. The reason for this decision is that Lindstrom et al’s method of subdivision would not allow the addition of many moveable positions unless 2 subdivisions were done, this is because the method on first subdivision creates extra positions on the edge of the square and none in the middle. See figures 16 and 17.

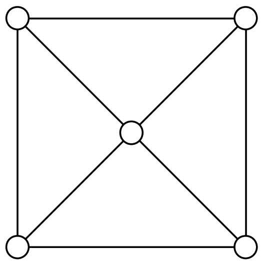
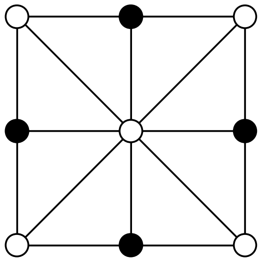
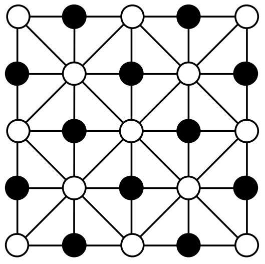




Figure 16, No Sub-divisions Figure 17. Figure 18.

Since the extra internal vertices and edge ones are needed for a significant increase in control over the terrain, creating 4 smaller squares allows the creation of additional vertices internally and on the edges creating a useful increasing to the ability to define more detailed areas on the terrain. This effectively means that Figure 16 is skipped altogether and the program starts with Figure 17. This is also because since faces can either be in polygon mode (simply 4 polygons rendered) or tree mode, if the area has no subdivisions it does not require an initially un-subdivided tree which has to be sub-divided as well as changing the face type.

The reason for subdividing areas of the terrain is that it allows more detailed modifications to be made to the terrain, since there is an additional vertex in the centre of the original square that can then be moved. As well as this 4 other vertices are created, one in the middle of each of the edges of the larger square. This can cause problems though as if these are moveable it becomes possible to create holes in the terrain See figure 18. Note since this problem was identified during the research stage it does not occur in the program, this image was created by temporarily removing the protection that is present. Lindstrom (1995) discuses this problem.

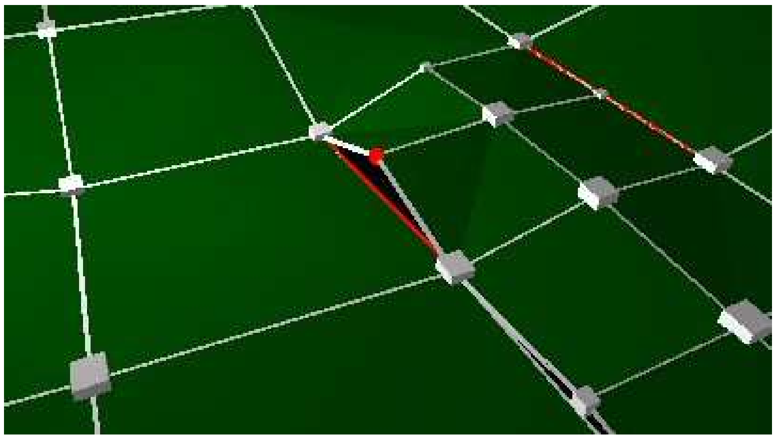


Figure 18, showing a hole in the terrain below the red square.

To prevent this the program initially creates the edge vertices as non-moveable vertices, these vertices are automatically adjusted by taking the average of the 2 end points of the edge, the same way as when they are initially made. This is done when the program displays the scene to ensure that moving any vertices does not create holes and inconsistencies. Obviously if 2 neighbouring trees are subdivided then it becomes necessary to be able to move these points as moving them will not cause holes. To solve this problem the program uses a lookup algorithm to determine whether a neighbouring tree has already been subdivided. Because of the pre planned structure of the trees, it is mathematically possible to determine which neighbouring tree needs to be looked at and which subdivision and in turn which vertex within that tree needs to be looked for to see if creation of a new vertex is needed. The reason for this check is that if the neighbouring sub-tree has already been subdivided, there will already be a vertex in the position that one of the vertices needs to be created in. If such as vertex is found, then the program uses the index of this vertex instead of creating a new vertex. This not only saves memory, but also allows the terrain to maintain consistency. One further process decision is also determined, if an existing vertex is found it is used in the creation of the sub-trees and the vertex is made moveable, if no vertex is found, then the program creates one in the position required and leaves the vertex as a non-moveable one to prevent inconsistencies.

To explain the algorithm completely is it required to understand further the labelling of the various sub-trees and vertices in the trees.

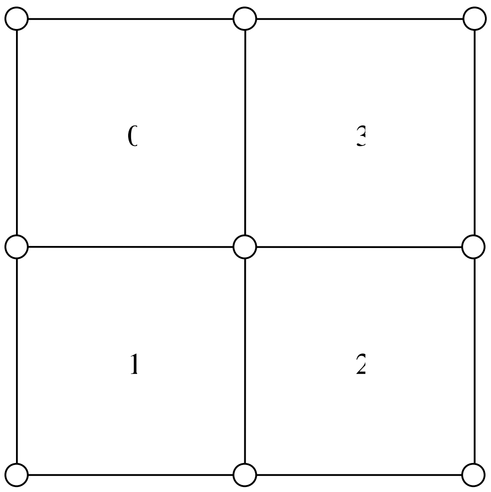


Figure 19

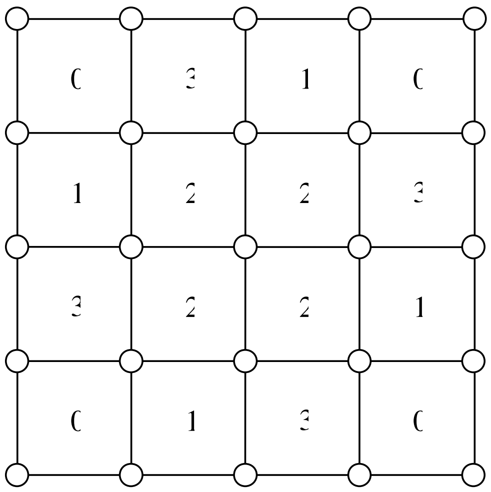


Figure 20

Figures 19 and 20 show how the sub-trees are identified, the corner sub-trees being names 0, then named incrementally in an anti-clockwise direction. Obviously the same principle of naming is used with the next level of sub-division also.

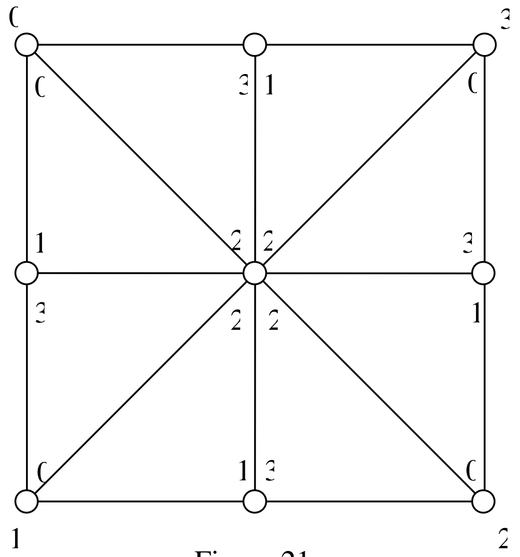


Figure 21

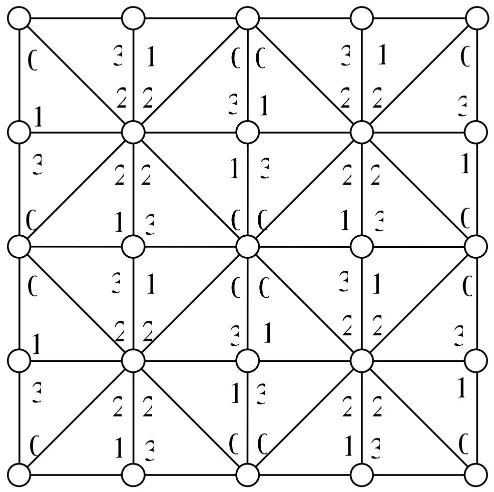


Figure 22

Figures 21 and 22 show the numbering convention for the vertices for each sub-tree. Like with the sub-trees the corner items are names 0, so the corner vertices are numbered 0, then named incrementally in an anti-clockwise direction. As before the convention is again used for the next level of subdivision.

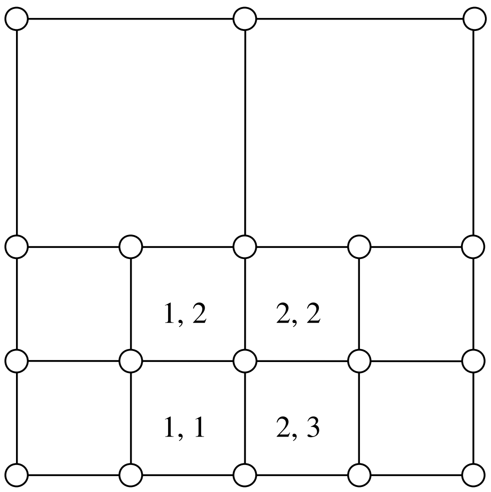


Figure 23

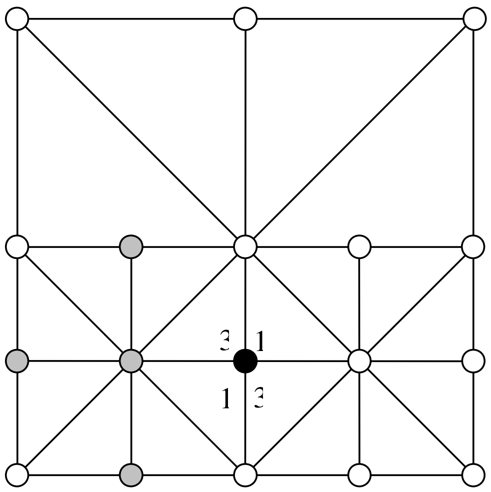


Figure 24

As it can be seen, sub-trees 1 and 2 will need to share a node when subdivided see figure 24, for sub-tree 1, the vertex that is shared is vertex 1 of sub-tree 1 that needs to be created, for sub-tree 2, the shared vertex is vertex 3 of sub-tree 3 that would be created. So there are two paths that would return the same vertex, 1 1 1 and 2 3 3 (figures 23 and 24 show this). Due to this it is clear that when looking for an existing vertex if the sub-tree anti-clockwise from the one being subdivided is needed, then 1 is added to the initial path number, then the rest of the numbers are simply complements of each other, i.e. 1's complement is 3, 2's is 2, 3's is 1 and obviously 0 is 0. So when sub-dividing sub-tree 1, a path of 2 3 3 is used for one border and 0 1 1 for the other border, when sub-dividing sub-tree 2 a path of 3 3 3 is used for one border and 1 1 1 is used for the other border.

Up to now the trees have all been completely separate from each other, the top most level of the trees share the vertices as they should, however the lower levels only share vertices if it requires looking inside the currant tree. If sub-dividing a tree will require making vertices on the edge of the root tree then a slightly different process is required. This requires looking in a different root tree from the one that the sub-trees being created are in. So as well as looking in a path like described in the previous section it is required to pass the root tree that needs to be looked in. This allows the separate root trees to be joined together correctly to create a fully modifiable net of vertices that can be manipulated.

As well as the functions described there are many others present in the Tree class, the following is a list of functions and a brief explanation of their purpose.

- Default constructor to create simple trees, note this should always be used in conjunction with methods to set the data up.
- A constructor to allow all the 4 vertices of the tree to be set as well.
- getType to allow the currant state of a sub-tree to be found, e.g. is the position specified a leaf or node.
- getTextureType to allow the easy retrieval of the currant texture for a specific sub-node.
- setTextureType to allow the easy setting of the texture for a specific sub-node.
- nodeToLeaf, this function converts a position to a leaf instead of a node, this actually only requires changing the mode, however it takes four argument which are the vertices to use for the corners of the leaf
- numberOfUses is a helper function to determine how many times a specific vertex has been used within the currant tree, this is used to assist the deletion of unused vertices.
- moveFacePositions, this is required if a vertex is removed, because the index values of all vertices in the vector array are changed, the trees need to be updated to make sure they use the correct points. This basically does a depth first search through the tree checking if any vertex references are equal or greater than the removed one, if one is found it reduces the stored value by one.
- removeVertex, this removes the vertex with a specific index value.
- makeVertices. This is perhaps the most complicated of the functions in combination with the next function, this performs the lengthy procedure described earlier, where neighbouring trees are checked for the presence of vertices that need to be used by this tree. It creates vertices if required or passes a reference to an existing one if suitable.
- leafToNode, this works in combination with the previous function, first the type of the position specified is set to be NODE, then the program uses the makeVertices function to create any extra vertices required, then it uses a temporary array created by that function to determine what vertices need to be stored in each of its sub-trees, finally it performs a nodeToLeaf operation passing the required vertices on each of the positions, sub-trees, in order to create the 4 sub-trees required. Finally the texture of each sub-tree is set to be the next texture sequentially from the first.
- getNode allows an entire sub-tree to be retrieved, this is used when trees need to be copied or searched etc.
- setNode is used to write back a modified tree into the required tree.

- getVertex returns the index number to use to find the vertex required, it does not return the actual TreeVector, just the index value.
- saveTree and loadTree, obviously are used for loading and saving the tree from/to a file. This uses a depth first approach to traverse the tree and then writes all the values to the file.
- drawTree is the function that renders the tree, as with many other operations it uses a depth-first traversal to look at the tree, if a node is found the function is recursively run with that sub-tree, if a leaf is found, it renders the polygons that are created using the vertex data stored for that leaf.
- findVertices is another helper function but is used for the collision detection system. This basically allows the program to determine all the vertices that are used by the currant tree to render the face. The collision detection requires this so that it can calculate what faces nearby need to be checked for collisions.
- The final function is calculateCost. This crudely calculates the area covered by the various leaves in the tree and determines how much the face costs based on the texture it is assigned.

3.3.2.4) Objects on the terrain

Objects can only be placed on vertices, the reason for this is to ensure that they do not pass beneath the surface of the terrain or hover over it. If objects need to be placed on an area where there is no vertex, then it is required to first increase the level of detail in the area required and then use one of the created vertices, Obviously although some vertices cannot be moved, it is possible to place objects on any vertex, as the positions of un-movable ones are still automatically adjusted and placing objects on them would not cause problems. The objects on the terrain using a struct called terrainObject, this simply stores the index number of the vertex it has been placed on, the type of object and the red, green and blue colour components of the object. The display function then chooses the appropriate drawing function to use depending on what object type is present and draws it using the colour stored and on the vertex stored. Since terrain objects require no actual specific functions on them as struct was used instead of a class.

Clearly it is noticeable that displaying the terrain is far more complicated than displaying the objects, this is the reason that although a larger number of objects would be better the project has not concentrated on including large numbers of different placeable objects, such as different size trees, different bushes and various other objects such as bridges and buildings. This would be a further expansion on the project.

3.3.2.5) Selecting objects

Object selection is done through the use of glPushName() and glPopName() to give different objects different names. The first name given to any object is it's type of name, this allows the program to know what kind of operation is required, and also to disallow running operations of the wrong type of selection, e.g. increasing the level of detail if an actually vertex was selected. There are 3 selection types, Vertex, Polygon and Qtree. For vertices only one more name is given and this is the number of the vertex, e.g. 0 if it is the first vertex in the vector array of vertices, etc. This allows the function that processes the hits from selection mode to know what vertex was selected, this is the only data required for all vertex operations. Polygons contain the same information as vertices, the fact that they are a polygon, then the polygon number which correlates to the number of the face used to define the polygon. It also contains one extra piece of information, which is the specific triangle within the face that is drawn, this allows the texture operations to know which triangles texture needs to be changed. The tree is far more detailed, it can contain several names, the first is the fact that it is a tree, the second name again is the tree number, corresponding to the face on the grid it represents, after this the names are the various sub-trees, the terrain can be sub-divided to a depth of 3 divisions, so in total a tree can have 5 names. Many of the functions support more sub-divisions than used, however only 3 seemed required and in fact any more make it difficult to see what it happening so a limit of 3 was chosen on the number of sub-divisions possible. Since objects only need a reference to the vertex they are placed on they use the same names as the vertex, this also allows selecting a tree to perform an operation on the vertex associated with it. This is not a problem, because the action will still determine what is done and so it is not be possible to perform the wrong option, e.g. remove a tree when a vertex is selected, or raise a tree instead of the vertex. In the case of raising and lowering points when selecting a tree raising the vertex is what is required anyway.

The function that processes hits obviously uses the name path and the currant action to determine the correct course of action, so no action can be made on an incompatible object selection. To further aid the user, when changing the action to be performed it automatically adjust the displaying of objects and points so that if an action can only be applied to a face it hides the vertices so that the user can not accidentally select a vertex instead of a face. This can be overridden if need be by using the tick box at the bottom of the screen.

3.3.2.6) The GUI

Initially all the controls were on a mouse menu, which was brought up by pressing the right mouse button, this menu system was very simple, however the menu was not user friendly, as a result a more graphical approach was adopted. The GLUI API was used to implement a

graphical interface; it works using functions compatible with GLUT and so maintains platform independence. The API supports buttons, number rollers, text boxes and various other objects. These were used to implement the controls to perform the different actions. Buttons are used to select action, then clicking on the terrain performs the action. Some controls, e.g. setting the texture require selecting an option from a drop down box, then clicking a face, others such as
Raising and lowering points also have a cursor area which can be selected using a number roller before performing the action.

3.3.2.7) Program functionality

There are several operations available to the user to create golf hole. The first action available is details, this gives the details of the object selected, any further details, such as the vertices being used by faces, or the co-ordinates of the vertex. This is mostly for determining the height of a point. But is also there for general information to the user. The next group of commands are for file operations, the group contains New, save and load, these obviously allow the user to start with a fresh terrain, save the currant terrain to a file, or load a terrain from a file.
The next group is the detail menu, this has 4 commands, increase detail, decrease detail, set texture and a drop down list to select the texture. These commands allow the user to adjust the detail of the terrain to make more detailed areas for the green etc. and also set texture and select texture, this allows the user to define what areas on the terrain are parts of the green, fairway, rough and other objects.
The following group is the vertex menu. This has just 2 commands, however there are other controls that are used with these operations. The commands are raise vertex and lower vertex, these allow the user to adjust the height of the points on the terrain. These are used in conjunction with the cursor area to determine how many vertices should be moved. Vertices are coloured to make it easy to determine what will be moved, with varying colour to help determine the impact of the operation. Figure 25, shows the area to be affected, figure 26 shows the result of performing the action.

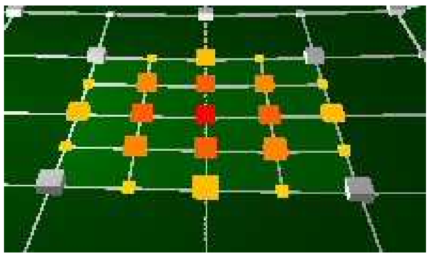


Figure 25.

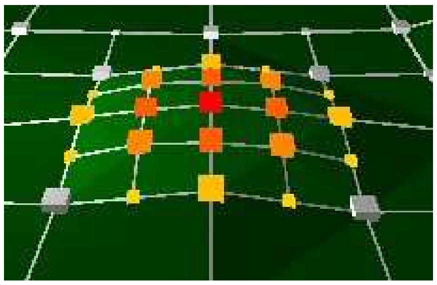


Figure 26.

The next group is the object menu. This has the commands relating to objects, firstly a drop-down list of colours is present in order to select the colour for objects such as flags and tees, then there are 3 commands to place objects on the terrain, place tree, place flag and place tee. Place flag and tee simply place an object of the desired colour on the terrain. Place tree works slightly different. If a face is selected it uses the cursor area to determine how many sub-divisions in the tree should be created and then places a tree on all vertices in the face, this only works when selecting a polygon, not a Qtree. If a vertex is selected then a similar gauge is provided as was provided with the raise or lower area command. See figures 27 and 28.

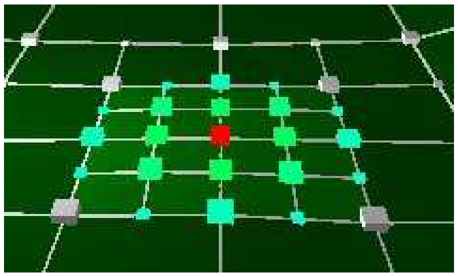


Figure 27.

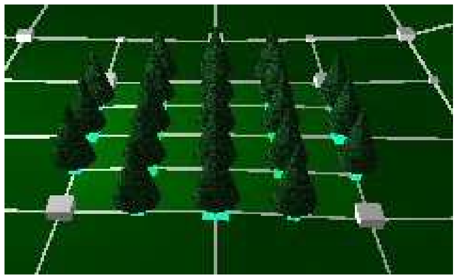
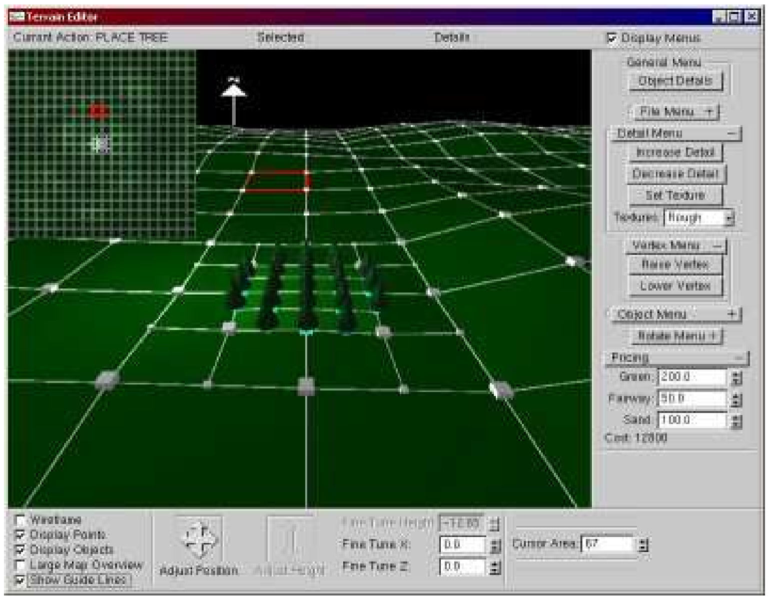


Figure 28.

The last command in the section is the remove object command, this basically uses the cursor area the same as for placing trees, but instead removes any objects on the coloured vertices. There is one more important functionality issue to point out with these commands that use the cursor area, since it is designed to show the user what will happen, initially clicking a vertex will simply move the cursor display to show the effects of the operation being performed on the newly selected vertex, to actually perform the task the same vertex must be selected again, effectively it works on the principle of preview then complete.
The next menu is the rotation menu, this allows the user to rotate the view by clicking and dragging the trackball, also a reset view is provided in case the user becomes disorientated. Note the compass does not respond to changes made using the command, as a result it is only really designed for demonstration purposes, as it allows the user to set the terrain rotating around a point without having to hold a key.
The final group is the pricing menu, this is simply 3 number rollers that can be used to enter the cost of the green, fairway and sand per square meter, and then the total cost is displayed below them. This can be used for very crude cost calculation.

There are other features on the screen, firstly is the mini-map, this is used to get an overview of the terrain so far, but also for quick navigation, clicking on the mini-map will shift the viewports focus to the selected face, and identifies the currently selected face by drawing a red box round it. As well as this there are several selection boxes at the bottom of the screen, these allow the program to draw in wire frame mode. Also whether points should be displayed or not, whether objects should be displayed, whether the mini-map should be made large or not See figures 29 and 30. Lastly there is a tick box to toggle the rendering of guidelines, these are the grey lines that separate the various faces.

Along the bottom is also the controls to manipulate the position of a vertex, either the drag controls can be used or exact values can be entered. The drag controls allow quick modification to the height or position of a vertex. Note the height options are only available if the currant action is either raise or lower vertex.



Current Action: PLACE TREE

[✓] Display Menus

Textures: Rough

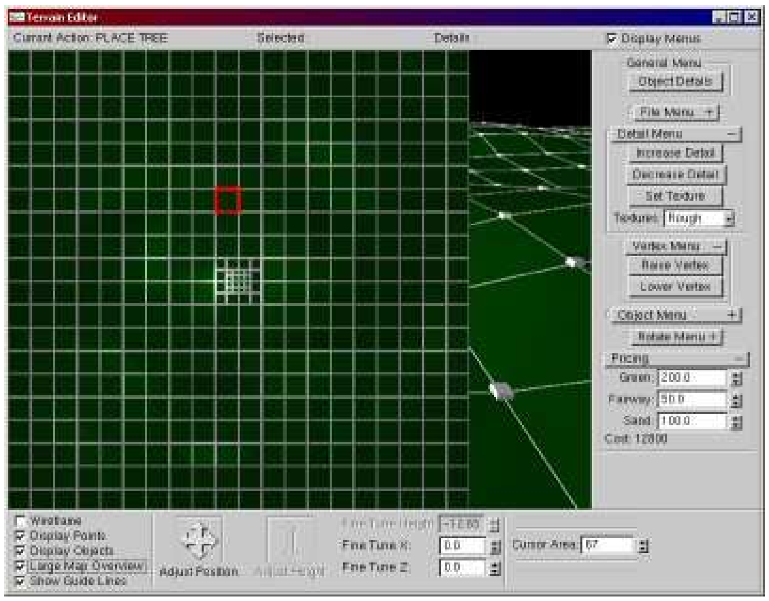
Green: 200.0
Fairway: 50.0
Sand: 100.0
Cost: 12800

[ ] Wireframe
[✓] Display Points
[✓] Display Objects
[ ] Large Map Overview
[✓] Show Guide Lines

Fine Tune Height: -12.80
Fine Tune X: 0.0
Fine Tune Z: 0.0

Cursor Area: 67

Figure 29, showing full interface.



Current Action: PLACE TREE

☑ Display Menus

Texture: Rough

Green: 200.0
Fairway: 50.0
Sand: 100.0
Cost: 12800

☐ Wireframe
☑ Display Points
☑ Display Objects
☑ Large Map Overview
☑ Show Guide Lines

Fine Tune Height: -12.65
Fine Tune X: 0.0
Fine Tune Z: 0.0

Cursor Area: 67

Figure 30, showing large mini-map

Figure 29, shows the finished program at time of final evaluation.

4) Evaluation

First in order to evaluate the program a set of criteria needs to be specified to use for evaluating the program. The aspects that can be evaluated can be split into two groups. These criteria are based on the functional and usability requirements that were defined in section 3.1.

4.1) The evaluation criteria

Firstly there are several areas of functionality that can be evaluated theses are:

- The ability to control the shape of the terrain.
    - F,a,i) Can points be moved on the terrain?
    - F,a,ii) Can areas of points be moved?
    - F,a,iii) Can the detail of the terrain be changed?
    - F,a,iv) Can detail of large areas be changed?
    - F,a,v) Is there good control over the level of detail?
    - F,a,vi) Can points be moved to any 3D position?
- The ability to specify which areas of the terrain belong to features of the course e.g. which areas are parts of the green, fairway etc.
    - F,b,i) Can areas be specified on the terrain?
    - F,b,ii) Can this be done for large areas?
    - F,b,iii) Can the user fully define the shape of the features? (e.g. curves).
    - F,b,iv) Can user control exact position of features on the terrain?
- The ability to place objects on the terrain and how flexibly the function works.
    - F,c,i) Can user place objects on the terrain?
    - F,c,ii) Can this be done for large areas?
    - F,c,iii) Can the user control the exact position of the objects?
    - F,c,iv) Can the user control aspects of the placed objects? E.g. colour and size.
    - F,c,v) Can a selection of objects be placed on the terrain, e.g. different types of trees, plants and bridges etc.
    - F,c,vi) Can these objects be removed?
    - F,c,vii) Can this be done for large areas?
- The ability to visualize the terrain.
    - F,d,i) Can the user easily visualise the terrain from a real life perspective?
    - F,d,ii) Can the user easily view specific areas of the terrain?
    - F,d,iii) Are small differences in terrain elevation easy to make out?

Secondly there are also several areas of usability that can also be evaluated.

- How easy the program is to use as an overall program.
    - U,a,i) Can appropriate functions be found easily?
    - U,a,ii) Can the user make a course without having significant problems?
- How predictable the behaviour of the program is.
    - U,b,i) Do commands behave in a manner that is constant?
    - U,b,ii) Do commands behave in a manner that is expected from the naming of them?
    - U,b,iii) Are all the commands obvious on their use?
- How well all the functional operations perform the task required.
    - U,c,i) Are the operations simple?
    - U,c,ii) Do the operations perform what is required?
- How small the learning curve is for the program.
    - U,d,i) Does the user understand the use of commands quickly?
    - U,d,ii) Can the use of commands to achieve the required effect be learned quickly?
- How easy are the functions to use.
    - U,e,i) Can the terrain be easily modified?
    - U,e,iii) Can features be easily defined?

<u>4.1) Justification for the evaluation criteria</u>

Obviously one big factor in the ability design a course is the ability to control the shape of the terrain, for this reason F,a,i) and F,a,iii) need to be checked to ensure the program meets the requirements. F,a,ii) and F,a,iv) are to evaluate how well the designer can manipulate large areas of the terrain. This is obviously important since the size of the terrain can be very large indeed. Just being able to control the level of detail on the terrain is not sufficient to state that the program can perform this satisfactory, the other thing that needs to be evaluated is the level of control available (F,a,v). The level of control over the points on the terrain also needs to be evaluated, can the points only be moved up and down, or is full control available (F,a,vi).

The first two F,b criteria are required since defining the green, fairway etc is essential to the creation of a golf course design. F,b,iii) was not initially part of the evaluation criteria, the reason it has been added is due to undertaking the evaluation. It has been added since it is something that should be evaluated based on the opinions of the evaluator. This is required since the designer should have control over the shape of features and not be limited to square

shapes. Another important fact is how flexible the positioning of features is, can they be placed anywhere on the terrain (F,b,iv).
Placing objects on the terrain is important as well since the flag and tee need to be specified, also trees need to be placed to define hazards etc. This is the reason for F,c,i). For objects such as trees, placement of only one tree at a time would be very time consuming so the ability to place areas of trees would be far more suitable (F,c,ii). A flexible program would allow the designer to place objects wherever require on the terrain (F,c,iii), also objects may need different characteristics, flags and tees require different colours so this needs to be possible, trees also need to be different sizes (F,c,iv). Lastly is a variety of objects provided for the user to place on the terrain, since a finished course may have many different types of items on it (F,c,v), it is important for the user to be able to place a good range of objects to make the design complete, this may even require placement of simple buildings to simulate the location of the club house etc. Obviously as well as placement of objects removal is also required so this needs to be evaluated (F,c,vi and F,c,vii).
The ability for the design to be visualised is important, firstly F,d,i) is required to see how well the user can visualise what the terrain will look like when it has been created in real life. The user should be able to easily view specific areas on the terrain (F,d,ii) this is because certain areas on the terrain will need more work than others, it should be simple to view different areas on the terrain. As well as being able to view different areas it is also important to be able to view the differences in the terrain (F,d,iii). Small changes in elevation can be very difficult to see in computer program, but may be very important in areas such as the green, where a very small change in height may have a very large impact to the required effect of the green's surface. Due to this, this is an important aspect to evaluate.

A good program will be easy to use, so the ability to find controls easily (U,a,i) and the ability to create a design without serious problems (U,a,ii) are very important.
A program will not be useable if a command does not behave consistently, since the user will be unable to determine what will happen if they use the command, due to this consistency (U,b,i) needs to be evaluated. As well as this commands should behave in a way that is expected from reading the name of it (U,b,ii). If a command increases the detail of an area of terrain it should be obvious from the name that this is what is done, e.g. calling such a function inc. d. would not be at all clear, whilst increase detail would be.
Any commands should work in such a way that the user knows what they are supposed to do with them (U,b,iii), this is loosely related to (U,b,ii) but concentrates more on the aspect of what should be done to perform a command, e.g. whether the command works on faces or vertices and how many items are affected.

Do the commands behave in a simple manner? (U,c,i) e.g. it is simple to do things, such as raise points, increase detail, change the area type etc.
Do the commands in the program perform the operation they need to perform completely (U,c,ii)? This needs to be evaluated since obviously if a command only works in certain circumstances the product does not fulfil all the requirements of it. An example of this may be only subdividing one square on the terrain and not all etc.
Do commands require large periods of time to learn (U,d,i)? A good program should be as quick to learn as possible, once a command has been used a few times it should not require reading help files to remember how to use the operation again.
The final aspect to evaluate is how well the user can determine what command is required to get the desired result (U,d,ii), if the user can not easily learn techniques required to get a desired result, more time will be spent trying to find out how to do something than actually doing it, this would obviously hinder instead of aid the design process.
The terrain should be easy to modify, this will be dependant on the functions that allow these changes to be made, U,e,i) is to determine how easily the terrain can be modified with the functions provided. The detail of the terrain will also be very important, how much control the program provides and how easy changes are to make is important. The evaluation will need to look at how easy it is to define different areas on the terrain (U,e,ii).

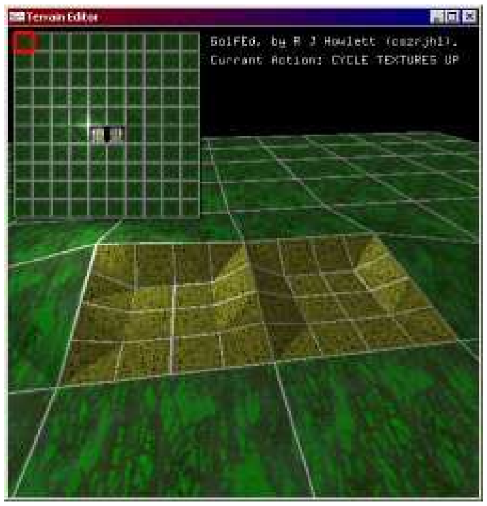
4.2) Performing the Evaluation

The evaluation was performed at several times, the first main evaluation was performed February 2004, then 2 further evaluations were performed by Peter Elmy of Elmy landscapes Ltd. during Easter 2004 one week apart. The various aspects of the program will be discussed, then a table will detail how well the program meets the criteria set. The program will be given 1 for criteria met, 0.5 for partially met criteria and 0 for criteria that has not been satisfied.
Elmy landscapes Ltd. Is a landscaping company that has worked on several golf courses in the East Anglia area, due to this the company seems to be a good place to look for evaluating a golf course design program.

4.2.1) Mid project evaluation

The mid project evaluation (February 2004), was undertaken by myself the author and Professor Ken Brodlie. The evaluation was to determine how the project was progressing and also identify possible improvements that are required. This version has most of the functionality required; however the general usability of the program requires work. The commands are all available from a menu that is brought up by using the right mouse button. This provides a very quick and easy interface for users that know what they are doing, however menu navigation can be confusing for users that do not have familiarity with the

program. The increase and decrease of resolution is simple to use and determined to be satisfactory to achieve the required effect the function is simple to use and provides good control over the terrain detail. The raising and lowering of vertices works well, however the function only affects single vertices, this is fine, however it makes it very difficult to move areas of vertices and still maintain smooth curved hills etc, some form of improvement was determined to be a good idea. This aspect of the program allows good control but is not useable on a large scale. The texture operations only allow changing the texture to the next available texture or the previous one. This has problems in that changing areas to all have the same texture can be rather time consuming, practically if all the faces in the area have different textures to each other, or if the texture required is several away from the currant one. The ability to specifically set a face to use a set texture would be far more useable and would also be far more efficient.
Currently the program only supports the placement of trees and the removal of trees, this obviously need to be changed so that at least flags and tees can be placed on the terrain. As well as this, placing large areas of trees is very time consuming as every vertex needs to be selected to place a tree on individually, some form of mass placement would be preferable. Another problem was identified with the behaviour of the mini-map due to the use of a 2D orthographic view, which causes raised areas to disappear on the mini-map. The largest functional flaw with the mid project evaluation program is that adjacent trees to not knit together see section 3.3.2.3 for details of this. This is a large problem because it greatly restricts where things can be created on the terrain, for example a bunker can only be made inside a single tree and can not overlap several different faces. See figure 31.



GolfEd, by R J Howlett (cmzrjhl).
Currant Action: CYCLE TEXTURES UP

Figure 31.

This was decided to be the greatest problem with the program at the time. Most of the other problems as mention relate to the requirement of some form of batch operations for commands, such as moving large areas of vertices or placing large areas of trees. This shows the ability to manipulate the terrain to be lacking and combined with the operations to move vertices and adjust the level of detail it is also not very user friendly at the moment.

criteria	score	criteria	score
F,a,i	1	F,c,vii	0
F,a,ii	0	F,d,i	0
F,a,iii	1	F,d,ii	1
F,a,iv	0	F,d,iii	0.5
F,a,v	0.5	U,a,i	0
F,a,vi	0	U,a,ii	0.5
F,b,i	1	U,b,i	1
F,b,ii	0	U,b,ii	0.5
F,b,iii	0	U,b,iii	0.5
F,b,iv	0	U,c,i	1
F,c,i	1	U,c,ii	0.5
F,c,ii	0	U,d,i	0.5
F,c,iii	0	U,d,ii	0.5
F,c,iv	0	U,e,i	1
F,c,v	0	U,e,ii	1
F,c,vi	1		
		Score of 31	14

4.2.2) Post mid-project evaluation changes made

One solution to the moving of large areas was to allow the user to define a path on the terrain, then use a new window to adjust the height of the terrain at the various selected points. This would then require the program to calculate the height of the other points along or near to the path in order to create a ridge or dip along the specified path. The same idea could be used as a method for sub-dividing the terrain around areas which need to be made into ridges so that the user does not need to manually increase the level of detail of every face that lies near the ridge being created. The other idea is to allow the user to specify an area of vertices to be affected around the vertex selected. This would require the user specifying a distance to affect, then the program would need to calculate the distance of all the vertices from the selected vertex and perform an operation on any vertices that are within the area of effect. This approach was selected as the method to use, the idea was implemented by adding an extra function to the TreeVector class. This function simply takes to TreeVector objects and calculates the distance between them. This is then used to calculate a weighted distance that specifies how far between the selected vertex and the furthest affected vertex would lie.

```
float weighted = ((cursorArea - vertices[vertex].distance(vertices[i])) / (float) cursorArea);
```

if the distance of a vertex from the selected one is 0, the weighted value comes out as 1, So the point will be fully affected. If the distance is equal to the cursorArea (the distance of the furthest vertex to affect) then the weighted value comes out as 0, meaning it should not be affected at all. Obviously a vertex half the distance of the cursorArea will yield a result of 0.5. Once a weighted value is found, the square root the weight is taken and multiplied by the distance to move the vertices by to calculate how much to move each vertex by. The reason for this is so that the closer to the selected vertex the other vertices are, the more they will be moved, the further away, the less they are moved. This allows the program to automatically make a smooth curve whenever an area of vertices is raised or lowered. This simplifies the task of maintaining a smooth terrain. This calculated distance is then added or subtracted from the original vertex position for each vertex affected. Initially this proved to be rather tricky to predict, since it was hard to tell what points would be moved how much. As a result the program was altered so that whenever the currant action was set to raise or lower, the program colours all the vertices that will be affected by the action using the same weighting function as before to alter the colour of the point. The program also uses the distance function of the TreeVector to determine an area for placing trees as well. As well as this it is now possible to add flags and tees as required to make a complete course.
The other major change to the program was the addition of a graphical interface, as opposed to just a mouse menu. This is discussed in section 3.3.2.6.

4.2.3) Easter evaluation 1

Intermediate evaluation (Easter 2004) identified several short comings with the program, the most notable of these was the lack of curves on the terrain. The evaluator suggested that this was a significant problem as a large aid to design is being able to easily represent a design to the potential customer. The problem being that customers would not want to view a potential design and see square areas where there should be curved lines, many customers would even pose the question as to whether the area is supposed to be square or not. This obviously presents a hindrance to the design process. Another of the main short comings was the lack of available objects to place on the terrain coupled with the lack of control over these objects. This is mainly relevant to the fact that only one species of tree could be visualised on the terrain, and no general plant life but also the fact that all the trees and objects are uniform size. Both of these problem obviously relate to the presentation of the design to a non-designer, trees being a symbolic representation so that the area can be identified as having

trees on it, squares being symbolic of a rough area that would be part of the green, fairway etc none the less these are 2 problems the program has.
The final short coming identified through the evaluation was the difficulty of visualising what the course would look like from a real life perspective.

The ability to increase and decrease the level of detail was seen as a positive attribute of the program, allowing the designer good control over the terrain in areas required, without having to increase the detail in areas that would be unaffected. One large problem being the necessity to subdivide each section one at a time, which can be very time consuming when a large area needs to be refined to the maximum detail level. A function to automatically sub-divide all squares to a certain depth would be useful. The mini-map allowed easy navigation on the terrain, however orientation was a problem as it was hard to tell which way was the top of the map if the 3D view was rotated.
The load command has the problem in that the user is required to remember the name of the file they wish to load, this would be made far better if a drop down list could be created containing a list of all the saved terrains.
The raise and lower commands evaluated to be very good, allowing a varying area of points to be manipulated at the same time. The automatic creating of curved hills as opposed to pointed peaks was also a very good aspect of the program, simplifying the process of raising or lowering areas whilst maintaining a smooth terrain. The cursor area identifies any vertices that will affected by the operations, by colouring them (The colour is changes the further the object is from the selected object), this allows the user to visually see what area will be affected and also give an idea as too how greatly the points will be affected.
Being able to place areas of trees was also helpful, allowing entire faces to be filled with varying density or placing trees on an area of vertices. This uses the cursor area like with raising and lowering vertices.

criteria	score	criteria	score
F,a,i	1	F,c,vii	1
F,a,ii	1	F,d,i	0
F,a,iii	1	F,d,ii	1
F,a,iv	0	F,d,iii	0.5
F,a,v	0.5	U,a,i	1
F,a,vi	0	U,a,ii	0.5
F,b,i	1	U,b,i	1
F,b,ii	0	U,b,ii	1
F,b,iii	0	U,b,iii	0.5
F,b,iv	1	U,c,i	1
F,c,i	1	U,c,ii	1
F,c,ii	1	U,d,i	0.5

F,c,iii	0	U,d,ii	0.5
F,c,iv	0.5	U,e,i	1
F,c,v	0.5	U,e,ii	1
F,c,vi	1		
		Score of 31	21

4.2.4) Post Easter evaluation 1 changes made

Following the feedback from the intermediate evaluation several changes were made to the program in order to meet some of the short comings identified, The first change was to allow the vertices on the terrain to not only move on the y axis, but to also move along the x, z plane. This allows increased control over the shape of areas on the terrain such as the green. This change makes it possible to create very crude curves on the terrain by moving the various vertices around the edge of a feature.

Another change made was to add a compass so that the user can always identify which direction on the 3D view is the north (Top of the mini map). This is to allow much easier navigation.

The largest addition was a first person view mode, which allows the user to walk around the terrain. This has full terrain collision but no object collision. Collision with objects is not essential to showing how the course would look from a real life perspective.

The textures were also changed to be plain colour images rather than textured images to make it easier to tell what areas are parts of each of the different features, this was decided upon by personal observations that it can be unclear if different faces on the terrain are supposed to be the same texture or not, due to the fact that the texture is applied so that the corners of the squares are the corners of a texture, obviously if an area has been subdivided the squares are smaller and the same size texture is used for a smaller square. The differing scales between different areas made it unclear which textures were the same. One option was to alter the UV mapping of the textures so that the scale was consistent with all sized squares, however changing to plain textures instead seemed a far simpler approach, the only disadvantage being the slight loss of realistic look, but this seems insignificant against making it clear which areas are what.

A trackball was added to allow rotation of the terrain without using the keyboard and an option was added to the top menu bar to allow the other 2 menus to be hidden. The reason for this is to reduce the on screen clutter when the options are not required, but also to allow for easier demonstration of a design, without seeing the designer operations.

The final change made to the program is the addition of a very basic cost calculator. The calculator only calculates the cost of the surface types that have been placed on the terrain. It calculates an approximate area that is covered by the green texture, fairway texture and sand, then it takes the user defined costs of these surface types and calculates the total cost of the terrain.

4.2.5) Final Evaluation

After the changes had been made to the program it was again evaluated to see how well it met the requirements, obviously only the evaluation of the changes to the program need be mentioned here. The modification to allow curves to be created by moving points was seen as a definite improvement as it allows at least some shaping that was not possible at all before, however smoother curves would have been far preferred. Another problem with the change is that it is very fiddly to create nice looking curves, also the control that allows the vertices to be moved easily has a problem in that if the 3Dview is rotated, the control still moves the vertex in the same direction in the 3D environment, so if the view is rotated round to look from the opposite side of an object, moving the control forward will actually move the vertex back and the left and right would be inverted. Due to this a far more user friendly control would be a definite requirement for future work. The compass made navigation much easier, it allows the designer to orient the 3D view with the mini-map to more easily navigate around the map using the mini-map.

The evaluator also found the first person view very useful, giving a fairly good idea of what a course would look like if walked over by a person. This would be very useful for demonstrating a design to potential customers.

The change of textures made an improvement to the clarity of the design, making it easier to see which areas share the same texture; however the new textures can sometimes make things look less clear due to the lighting making areas with the same texture look different from each other.

Before being added the evaluator had already been asked whether a form of cost valuation would be useful, the idea was given a very positive feedback. Since this was added as a very basic cost calculator not a fully detailed one the evaluator gave the performance of the function a useful but needing improvement response. If fully implemented it would be very useful for assisting design, the idea being that if a design is shown to a customer and they wish to know the cost, it can be quickly found, then if the cost is too great the design can be very easily adjusted, e.g. by reducing the size of the green, removing some areas of added greenery or even reducing the amount of geometric changes made to the terrain. The later would obviously require tracking any changes made to the heights and positions of the vertices in the terrain, or at least comparing the designed course to the original terrain it was created on. This would definitely be an area to improve for future versions.

criteria	score	criteria	score
F,a,i	1	F,c,vii	1
F,a,ii	1	F,d,i	1
F,a,iii	1	F,d,ii	1
F,a,iv	0	F,d,iii	1
F,a,v	1	U,a,i	1
F,a,vi	1	U,a,ii	1
F,b,i	1	U,b,i	1
F,b,ii	0	U,b,ii	0.5
F,b,iii	0.5	U,b,iii	0.5
F,b,iv	1	U,c,i	1
F,c,i	1	U,c,ii	1
F,c,ii	1	U,d,i	0.5
F,c,iii	0.5	U,d,ii	0.5
F,c,iv	0.5	U,e,i	1
F,c,v	0	U,e,ii	1
F,c,vi	1		
		Score of 31	24.5

4.3) Conclusions drawn

The evaluation scores show that the project did progress towards meeting more design requirements as the project progressed, the initial score of 14, then 21, and finally 24.5 as percentages, these work out as 45%, 68% and 79% of the requirements met at these stages. Clearly some of the criteria would be unlikely to be met well due to the introduction of them at a later stage in the project. The results show that the prototyping and incremental development methodologies did facilitate the meeting of the requirements and proved to be an effective approach to the problem. Many of the other requirements not met would be easily met over time through continuation of the project. One thing of note is that some changes made between the 2 final versions of the program cause the program to only compile on windows. Up until this point the source code could compile on both UNIX and windows based systems, greatly adding to the products value since it does not require code changes to create copies for either system. A key change for future work would be to alter the first person camera code to use commands that are compatible with both systems.

There evaluation identified several key elements that should be improved upon, firstly is the ability to affect larger areas, this should be expanded to include sub-dividing mass areas as well and being able to define the number of levels to divide by as well. Setting textures for large areas would also be useful. The program would be far better if written to better support curved shapes, not just very crude approximations. The program would benefit from a system where the user can select any location on the terrain to place an object, as opposed to having to move a nearby vertex to the required potion, as this can cause large gaps between trees if one point is moved a long distance. This would require using the collision detection to determine the correct height for the objects to be placed at. Greater control over the objects placed on the terrain would be useful, originally when implementing colour control is was though to use separate user defined values for the red, green and blue components, this would be preferable but would have required some way of selecting common colours, the drop down list seemed a better solution at this time, this the implementation of full colour options added at a later date.

Control over the size of the objects would also be an advantage, allowing different sized trees to be placed etc, this could easily be added due to the implementation of the functions to draw objects taking scale values anyway. As well as this the addition of extra objects was always intended and would be another area to improve upon. General usability should always be worked on when expanding or revising any program.

There are also several other areas that could greatly be improved, one such area is the control over the features on the terrain. Although the currant approach is adequate, an approach of painting the texture on instead may have proved more appealing, this could involve some form of bitmap that is mapped onto the terrain as it's texture, which can be modified using a built in pain program. This would allow far more precise control over features, since they can be painted exactly where required. Obviously the costing feature would be good to complete also. Lastly some of the extensions could be added with future work, especially the ability to play the course with a built in golf simulator.

5) Bibliography

Bennet Simon, McRobb, Steve And Farmer, Ray (1999), Object-Oriented Systems Analysis and Design using UML, McGraw-Hill publishing company [05/05/2004].

Golf therapy retreat and wellness research centre (2001), Home Page,
URL: http://www.getupandgolf.com/teetime.php3 [20/11/2003].

Lindstrom, Peter, Koller, David, Ribarsky, William, Hodges, Larry F., Faust, Nick and Turner, Gregory A. (1996), Real-Time, Continuous Level of Detail Rendering of Height Fields, pp. 3-7. [20/04/2004]

Lindstrom, Peter, Koller, David, Ribarsky, William, Hodges, Larry F., Faust, Nick and Turner, Gregory A. (1995), Level-Of-Detail Management For Real-Time Rendering Of PHOTOTEXTURED Terrain, pp. 3-7. [20/04/2004]

Professional Golfer's Association (2003), Home Page,
URL: http://www.pga.com/learn/ [20/11/2003].

Taillon, Frederick (2003), Editor42 home page,
URL: http://bfed.3dmax.org/ [11/12/2003].

The American Society of Golf Course Architects (2003), Developers Page,
URL: http://www.golfdesign.org/public/connect/home.html?c=70120742&pageid=11577 [22/11/2003].

The University of British Columbia (1997, 1), Department of geography,
URL: http://www.geog.ubc.ca/courses/klink/gis.notes/ncgia/u38.html [9/12/2003].

The University of British Columbia (1997, 2), Department of geography,
URL: http://www.geog.ubc.ca/courses/klink/gis.notes/ncgia/u39.html [9/12/2003].

The University of British Columbia (1997, 3), Department of geography,
URL: http://www.geog.ubc.ca/courses/klink/gis.notes/ncgia/u36.html [9/12/2003].

6) Appendix A

The personal objectives of the project were mainly to look into more detail on computer based landscapes, although this was partially satisfied by looking at multi-resolution terrains, it did not fully satisfy the objective. It would have been preferable to have been able to concentrate more on the terrain aspect of the problem than the actual design aspect of the problem. This was one problem with the project in that the focus was lost at several points were work was being done that did not really satisfy the design requirements, for example the research based on terrain file storage which ultimately does not fulfil any requirements of the program but was completed through interest in the area of file formats for terrain storage.

On the whole the project went fairly well, Several issues of implementation would have been done differently if the opportunity arose. The main difference would be the use of a multi-resolution terrain. Although it is very useful the time required to implement this may have been far better spent trying to achieve some of the other requirements of the program. The first prototype program by itself was already able to achieve many of the requirements and could have been improved upon to meet the requirements far more quickly than the approach taken. Due to this an approach using the first prototype with a fixed resolution would be chosen, simply ensuring that the resolution is detailed enough for the most defined areas required on the course. Another error was the assumption that a design tool need only be based around functionality and the ability to use symbolic representation. In reality it is apparent that a key element to assisting design is also to make sure it looks good, due to this a very different approach would be taken to defining features. As suggested in the evaluation, a system of providing the user with a pain brush allowing them to define exactly where the features of the course are would have been a better approach to this part of the problem.

The main parts of the project that went well despite the time taken is the ability to have multiple resolutions on the terrain. The implementation worked very well and satisfied some personal objectives to experiment with terrains.

7) Appendix B – Borrowed code

Several parts of the program were taken from various tutorials available on OpenGL programming. Firstly the textures loading function was taken from the NEHE tutorials available at http://nehe.gamedev.net/counter.asp?file=files/basecode/nehegl_glut.zip
This is linked to from the main NEHE site at http://nehe.gamedev.net.

The code for the frustum culling and the collision detection functions were taken from www.GameTutorials.com. The collision detection code and first person camera code used comes from the Camera and World Collision tutorial available at http://www.gametutorials.com/download/OpenGL/CamWorldCollision_OGL.zip. The Frustum culling code was taken from the Frustum Culling tutorial available at http://www.gametutorials.com/download/OpenGL/FrustumCulling_OGL.zip.

Since these requirements had already been solved it is a complete waste of time creating a solution to a problem that has already been solved well. This is the reason that the code has been used. All culling and camera collision code that has been directly copied is present in the Frustum.h and Frustum.cpp files of the project and the texture loading code used is in the Tga.h file of the project.

8) Appendix C – User manual

This appendix is designed to offer an explanation of each command available to the user and how it should be used, as well as how to use the commands.

Initially there are several functions that are simply for assistance.

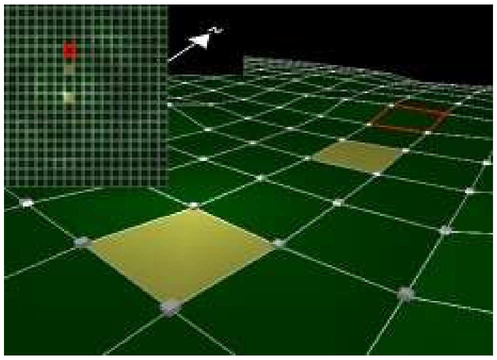


Figure A.

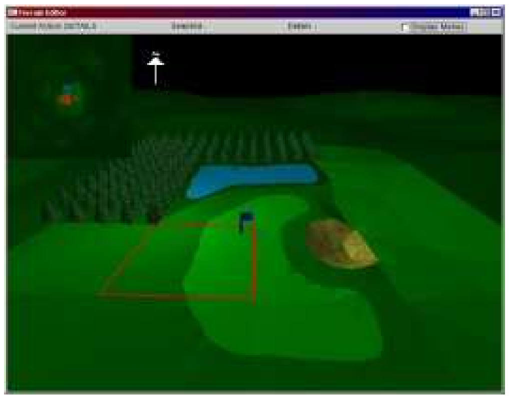


Figure B.

Figure A shows the compass that allows easy recognition of what orientation the map is at, the arrow points towards the direction on the 3D map that corresponds to the top of the mini-map.

Figure B shows the right and bottom menus being hidden by using the check box on the top menu bar.

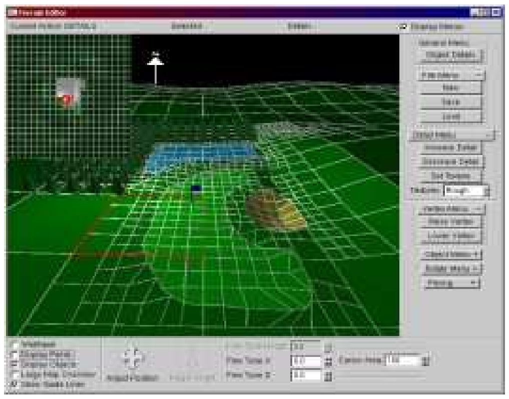


Figure C.

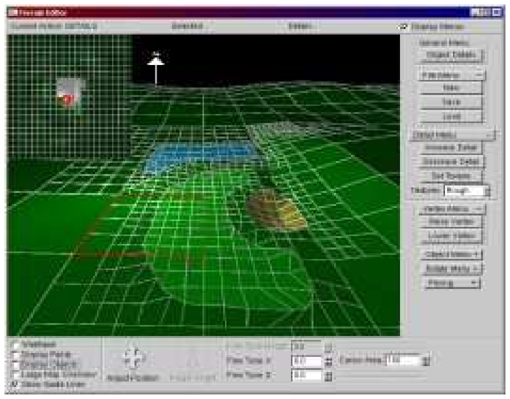


Figure D.

Figure C shows the terrain displayed without any points, figure D shows this but also has the displaying of objects turned off. These are done by clearing the checkboxes on the bottom panel.

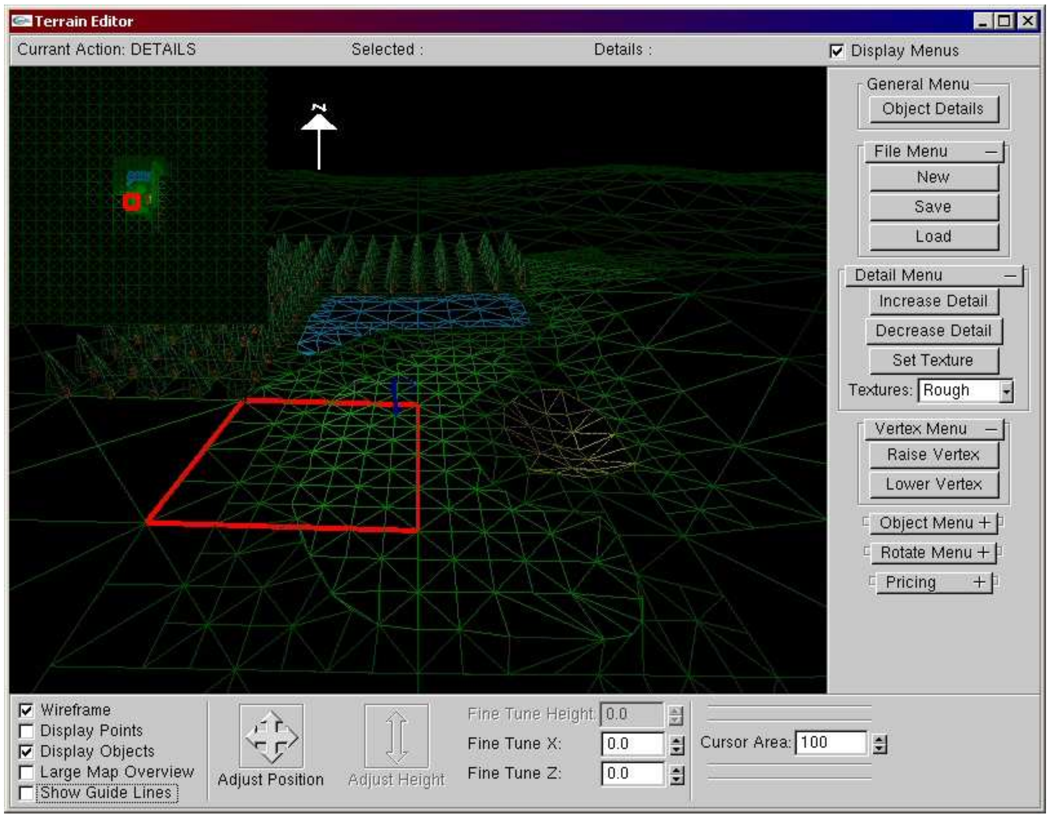


Figure E, showing the terrain displayed in wireframe mode instead of filled polygon mode, selected by checking the wireframe button on the bottom panel.

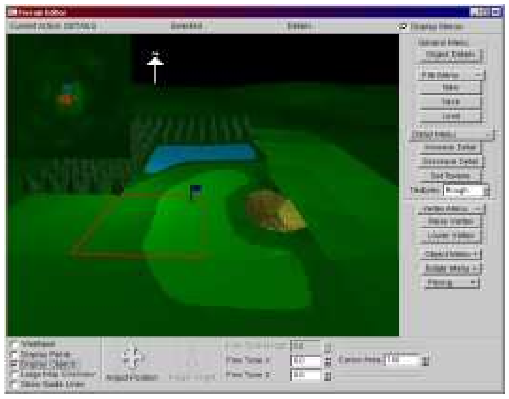


Figure F.

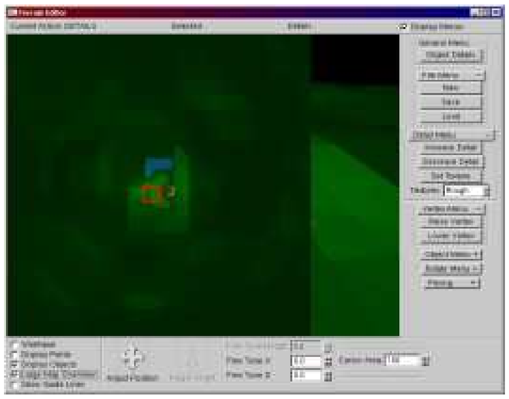


Figure G.

Figure F shows the terrain without the guide lines, this is done by clearing the checkbox on the bottom panel.

Figure G shows the mini-map in an enlarged state, this is to allow an overview of the map to be examined. This can be done by checking the large map checkbox on the bottom panel.

The Commands

The following section details the commands available and their use; firstly the detail commands will be discussed. These commands allow the user to specify the detail of the terrain.

Detail Menu

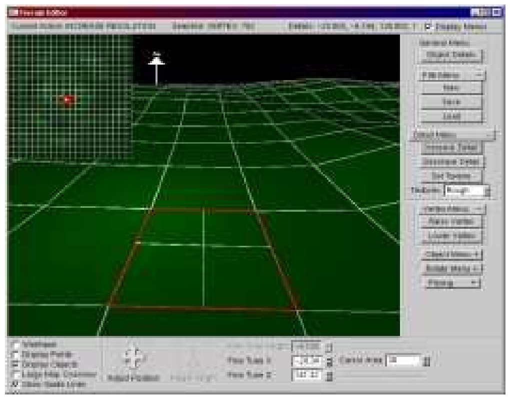


Figure H-i.

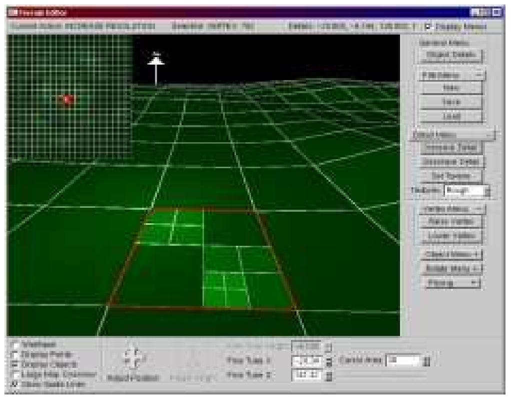


Figure H-ii.

Figure H-i shows one single face converted from polygon mode, to tree mode, this procedure also creates extra vertices and is required to increase the level of detail of an area. Figure H-ii shows subdivisions of some of the sub-trees of that face. To increase the resolution of the terrain, simply select the increase detail command from the detail menu on the right panel, and then simply click a square on the screen. The selected square will then be sub-divided into 4 smaller squares and depending on the neighbouring areas several vertices are also created to allow more control of the terrain.

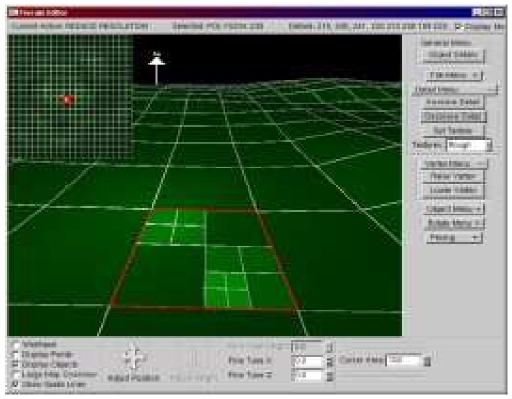


Figure I-i.

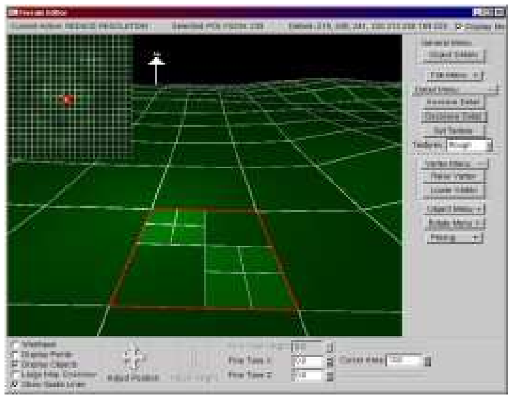


Figure I-ii.

Figures I-i and I-ii show the process of decreasing the resolution of an area, this command works the same way as increasing the resolution, however selecting a square will remove the square selected as well as the 3 other squares that make up the root square they belong to. Simply select decrease detail then click a square on the screen.

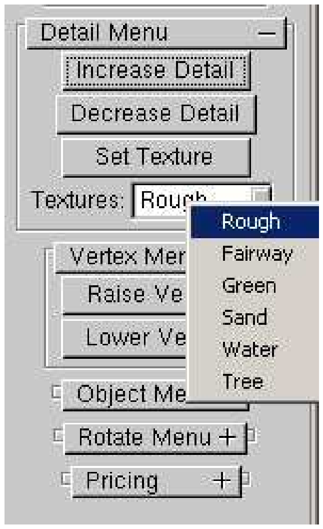


Figure J-i.

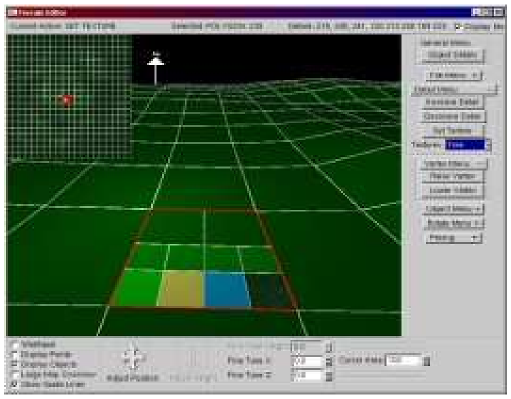


Figure J-ii.

To change the texture of an area on the terrain, select the drop down menu from the detail menu on the right panel and select the required texture (Figure J-i), select the set Texture command. Then click on a square on the terrain. Figure J-ii shows the available textures all placed on the terrain. A Tree texture has been provided so that for large areas of trees a texture can be used instead of placing many trees on the terrain. This is for performance reasons.

<u>Cursor Area</u>

Some of the following commands use a value called the cursor area; this defines the area of effect for several commands.

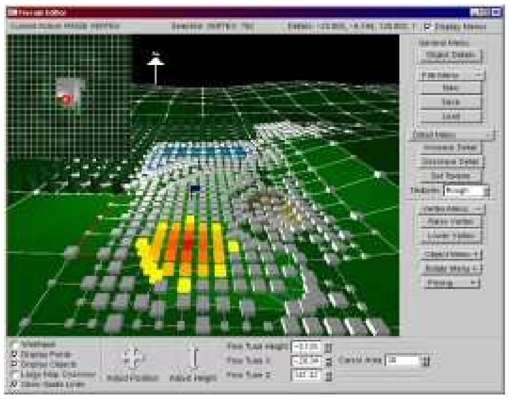


Figure K-i.

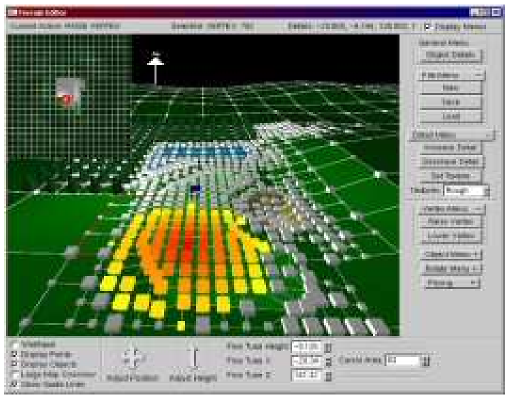


Figure K-ii.

Both figures K-i and K-ii show the areas that will be affected by the raise or lower area commands, identified by the red to yellow colour of the vertices. If the command is place or remove objects, then the vertices will be green coloured. The cursor area is changed by using the number roller on the right side of the bottom panel. As the area is changed the vertices that are coloured will change to reflect the new area of effect.

<u>Vertex Menu</u>

The vertex menu contains the commands required to adjust the height of objects on the terrain.

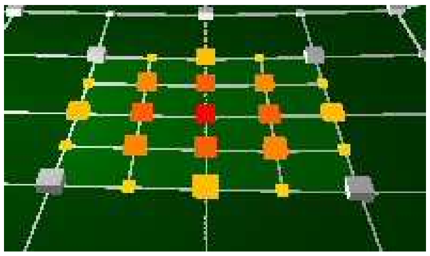


Figure L-i.

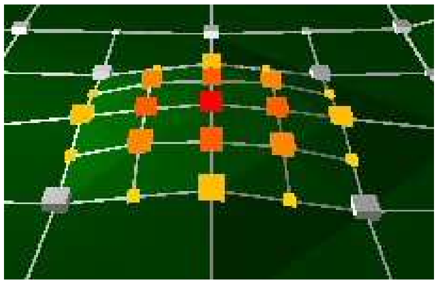


Figure L-ii.

Figures L-i and L-ii show the raise are command. When first clicking any vertex the program moves the focus point of the command (The red cube) and uses the cursor area to show the user what points will be affected. The area can then be raised in several ways; either a vertex can be repeatedly clicked increasing the height a small amount each time, the other option is to use the drag control on the bottom panel to drag the area up or down. The last approach is to actually specify the required height using the fine tune height box on the bottom panel. The raise and lower functions both behave identically. The command automatically creates curves if areas of vertices are affected.

<u>Object Menu</u>

This contains any commands required to place objects onto the terrain. This includes trees, flags and tees. It also provides the commands to remove objects from the terrain.

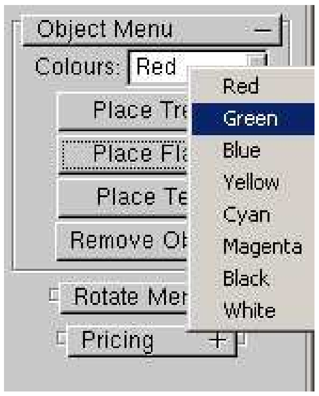
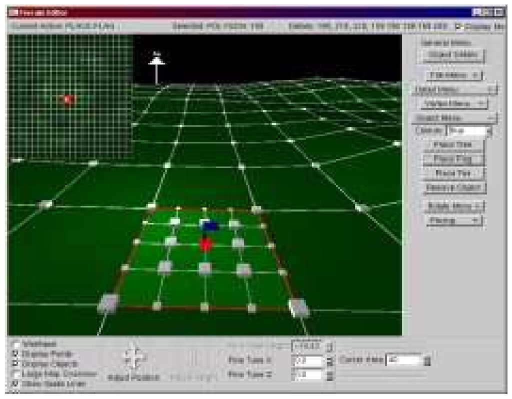


Figure M, showing the drop down list for selecting the colour of an object to be placed on the terrain.



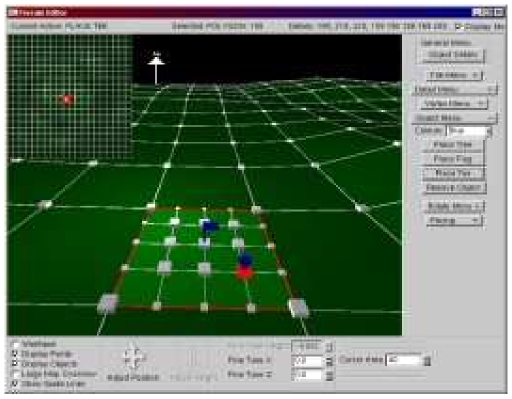
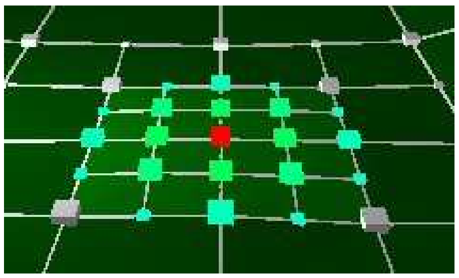


Figure N. Figure O.

To place a flag on the terrain simply select the colour of the flag required (Figure N) then select the place flag command from the menu then finally click on a vertex in the 3D view, this will place a flag on the terrain on the vertex selected (Figure O). The same procedure is used for placing a tee except obviously the place tee command must be selected.



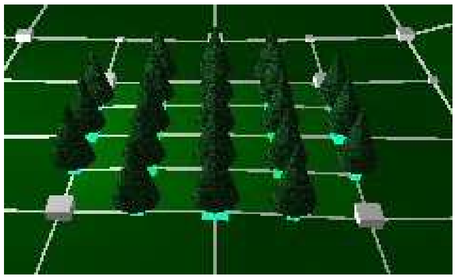


Figure P-i. Figure P-ii.

To place trees first select the place trees command, then click on the vertex where the central tree is required, now set the cursor area to the value required to determine the number of trees that need to be placed (Figure P-i). If the area needs to be moved, simply click on a different vertex, once the area of effect looks correct click on the currently selected vertex a second time. This will place a tree on any vertices that were coloured by the cursor identifier (Figure P-ii).

If a polygon is selected instead of a vertex the operation is slightly different.

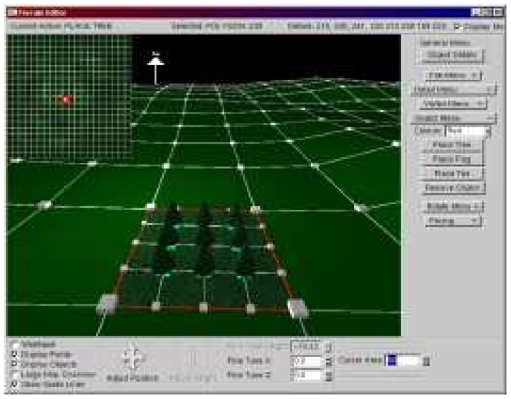


Figure P-iii.

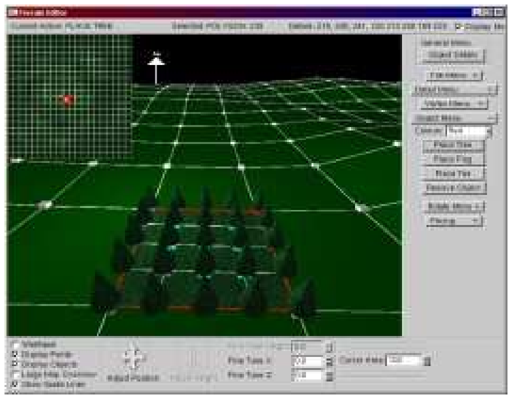


Figure P-iv.

The cursor area will instead determine the density with which vertices will be placed within the selected face, if a value under 50 is used, the face will be sub-divided once and a tree placed on the central vertex created. If a value between 49 and 100 is used the face will be sub-divided 2 levels (converted to a tree then sub-divided once) and trees placed on the 9 central positions created (Figure P-iii). If a value of 100 or greater is used then the face will be sub-divided 3 times (converted to tree then sub-divided twice) and trees are placed on all the created vertices (Figure P-iv).

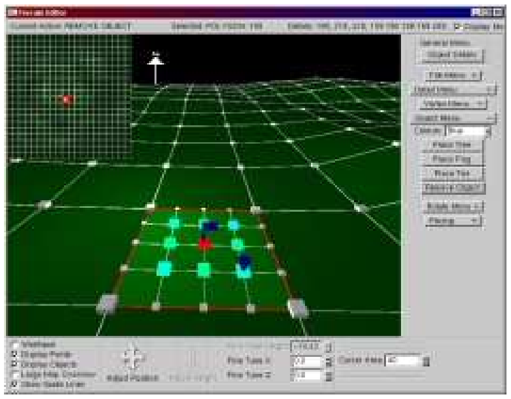


Figure Q-i.

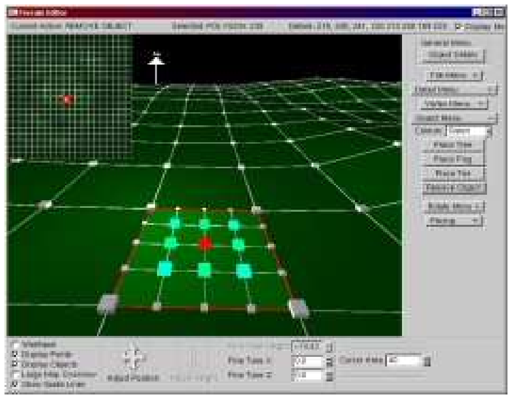


Figure Q-ii.

To remove objects the same procedure is used as with placing trees, select centre vertex, select cursor area (Figure Q-i) then click vertex again. This will remove objects from any vertices that are within the cursor area (Figure Q-ii).

<u>Rotate Menu</u>

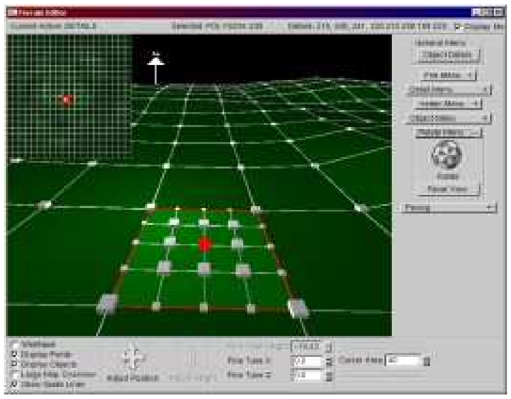
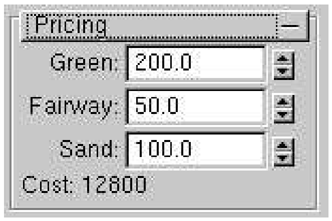


Figure R.

Figure R shows the rotate menu unrolled. The trackball can be used to rotate the view round, this is fiddly to use however can be used to rotate the terrain about the centre of the terrain. The other useful feature is the ability to set the terrain continuously rotating; this can be used to slowly rotate the view to show customers and other users. There is a reset command below the trackball that allows the view to be reset to the standard view.

<u>Pricing Menu</u>



Pricing

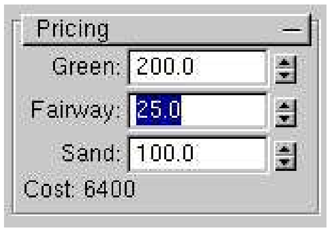
Green: 200.0

Fairway: 50.0

Sand: 100.0

Cost: 12800

Figure S-i.



Pricing

Green: 200.0

Fairway: 25.0

Sand: 100.0

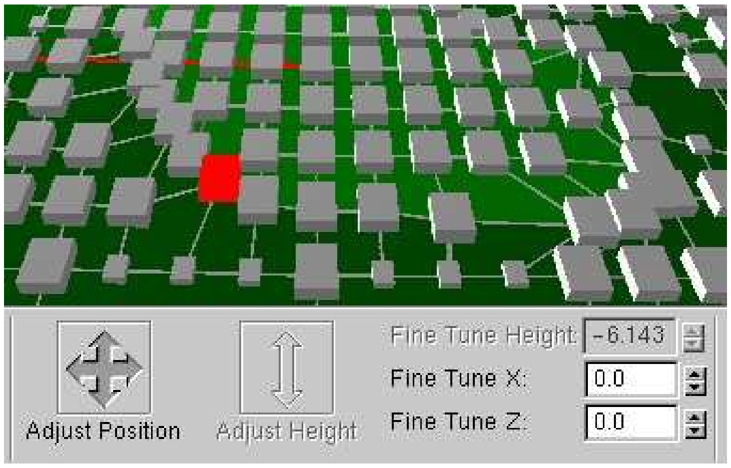
Cost: 6400

Figure S-ii.

The pricing of the terrain is done automatically; it calculates the amount of terrain that is covered by green, fairway and sand and uses the values in the menu to determine the cost of each type of terrain feature. Basically it will calculate the cost of the terrain types present in the design and add them together (Figure S-i). Figure S-ii shows the different cost value as a result of halving the cost of the fairway per square meter. The cost is also half because only fairway features were specified on the terrain. Note the initial values are only there for demonstration purposes, the designer would need to calculate their own costing for the different surface types and input the values.

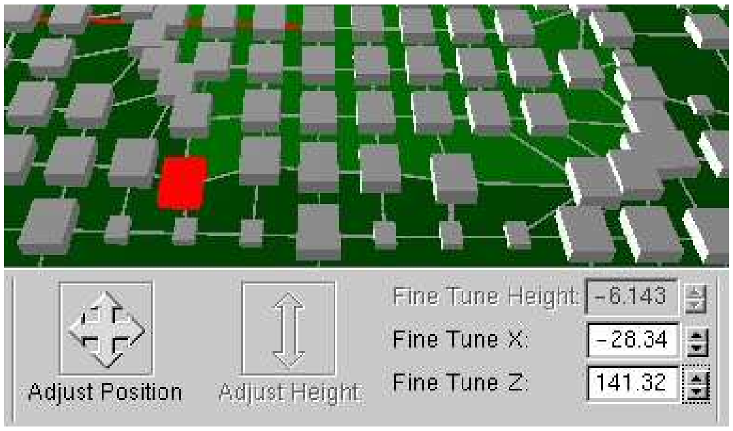
Moving Vertices

The vertices on the terrain can be moved in several ways the most important is the adjustment of the height, this is done through the use of commands discussed later. As well as this the position of the vertex on the plane can be adjusted. This is mainly for allowing features to be defined more precisely and with smoother curves rather than corners.



Fine Tune Height: -6.143
Fine Tune X: 0.0
Fine Tune Z: 0.0

Figure T-i.



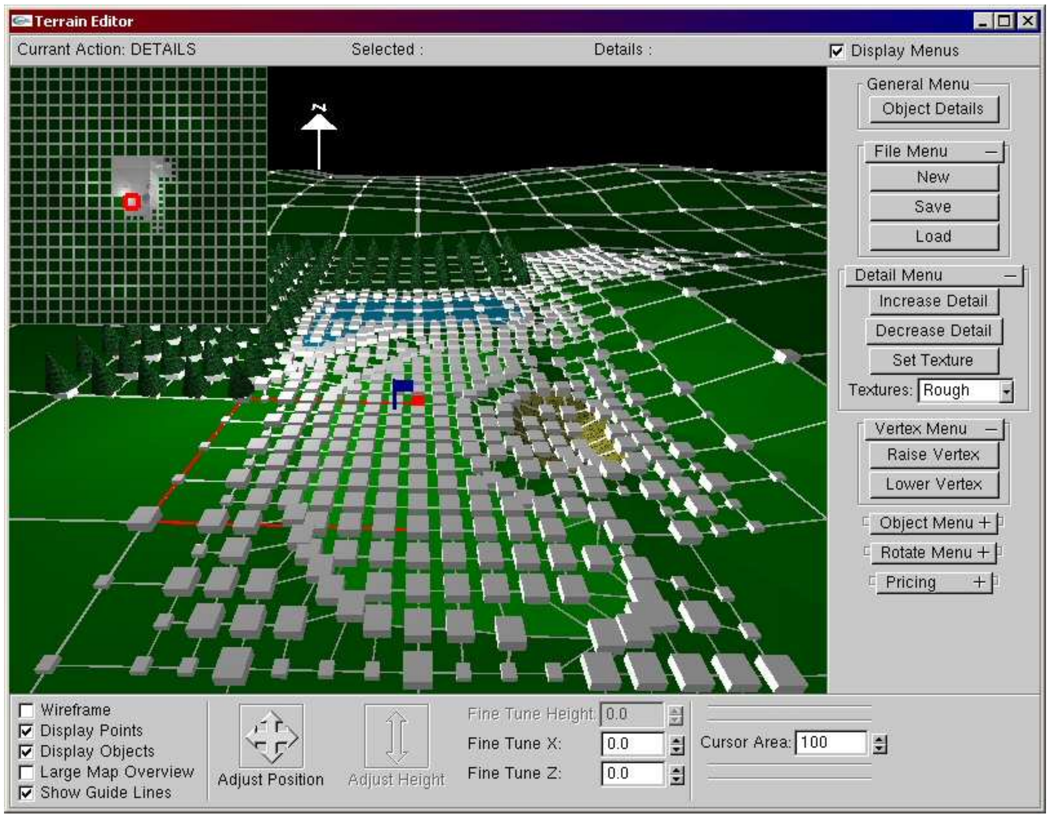
Fine Tune Height: -6.143
Fine Tune X: -28.34
Fine Tune Z: 141.32

Figure T-ii.

Figure T-i shows the position of a vertex and Figure T-ii after using the adjust position drag control. This control allows the vertex to be dragged to a new position. The position can also be moved by using the fine tune X and Z controls and writing in the required position.

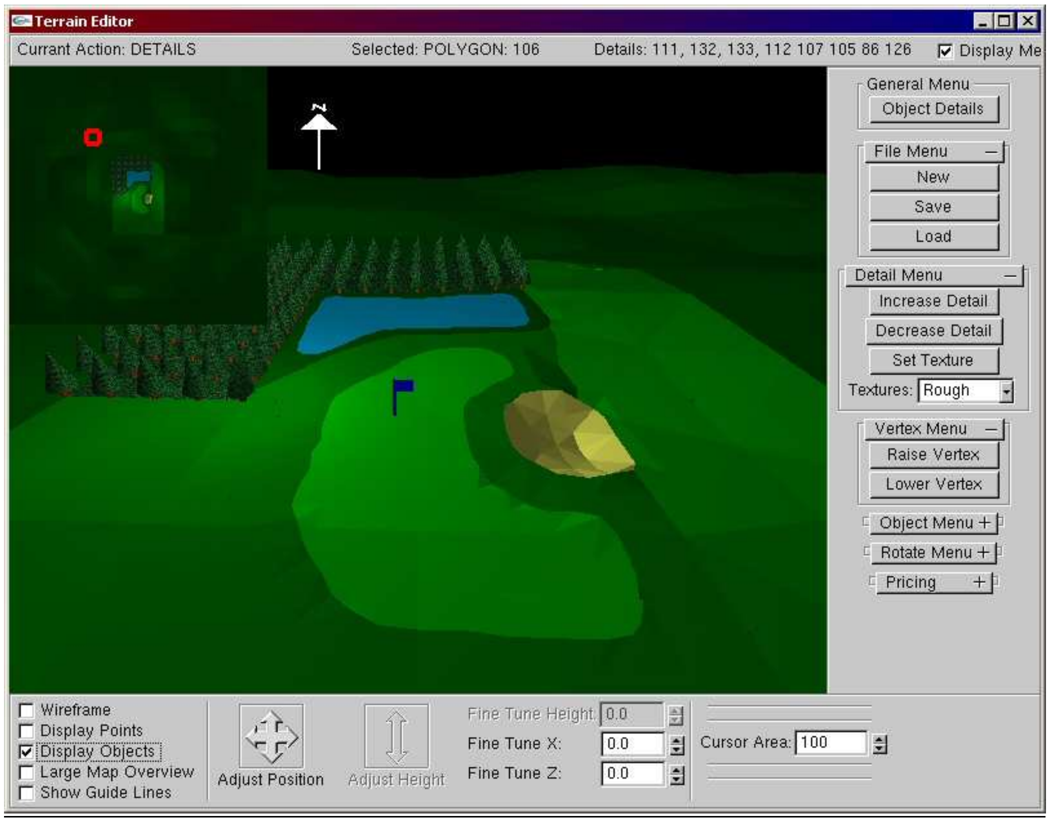
Product Features

The following screen shots show the product as a whole, Figures U-I and U-ii show a sample design created with the program.



Currant Action: DETAILS
Selected :
Details :
[✓] Display Menus
Textures: Rough
[ ] Wireframe
[✓] Display Points
[✓] Display Objects
[ ] Large Map Overview
[✓] Show Guide Lines
Fine Tune Height: 0.0
Fine Tune X: 0.0
Fine Tune Z: 0.0
Cursor Area: 100

Figure U-I, showing the designed terrain with points and guide lines.



Currant Action: DETAILS Selected: POLYGON: 106 Details: 111, 132, 133, 112 107 105 86 126 ☑ Display Me

Textures: Rough

☐ Wireframe
☐ Display Points
☑ Display Objects
☐ Large Map Overview
☐ Show Guide Lines

Fine Tune Height: 0.0
Fine Tune X: 0.0
Fine Tune Z: 0.0
Cursor Area: 100

Figure U-ii, the design from U-i without design aids showing.

One extra feature that has been added to the program is a first person view mode. This can be started by pressing A (Case sensitive). This mode can be left by pressing a (Case sensitive). This mode allows the user to walk over the terrain and see what the course would look like in a real life perspective. Figures V-i, V-ii and V-iii show some sample views from the first person mode.

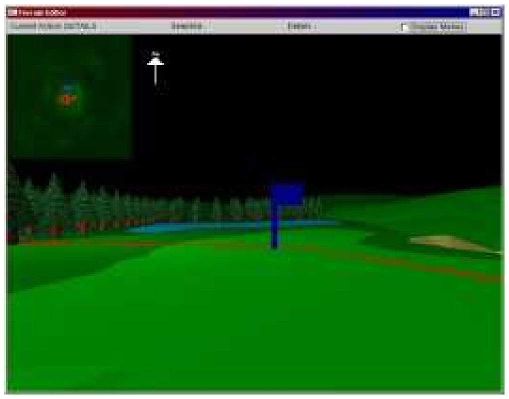


Figure V-i.

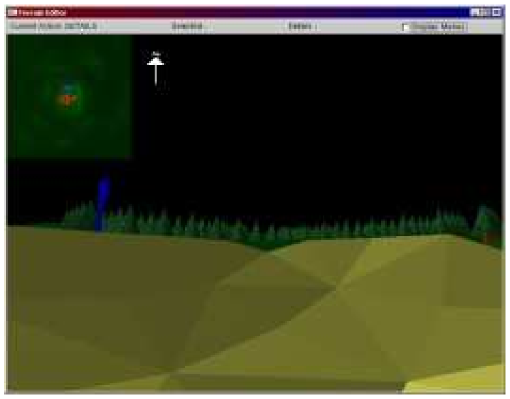


Figure V-ii.

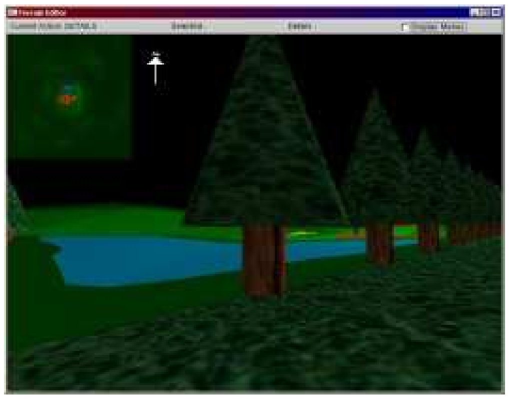


Figure V-iii.

When in first person view the keyboard controls are as follows, j moves left, l moves right, I moves forward and k moves backwards. Holding the shift key down while pressing any of these keys will cause the user to move more quickly over the terrain. The mouse is used to look around.