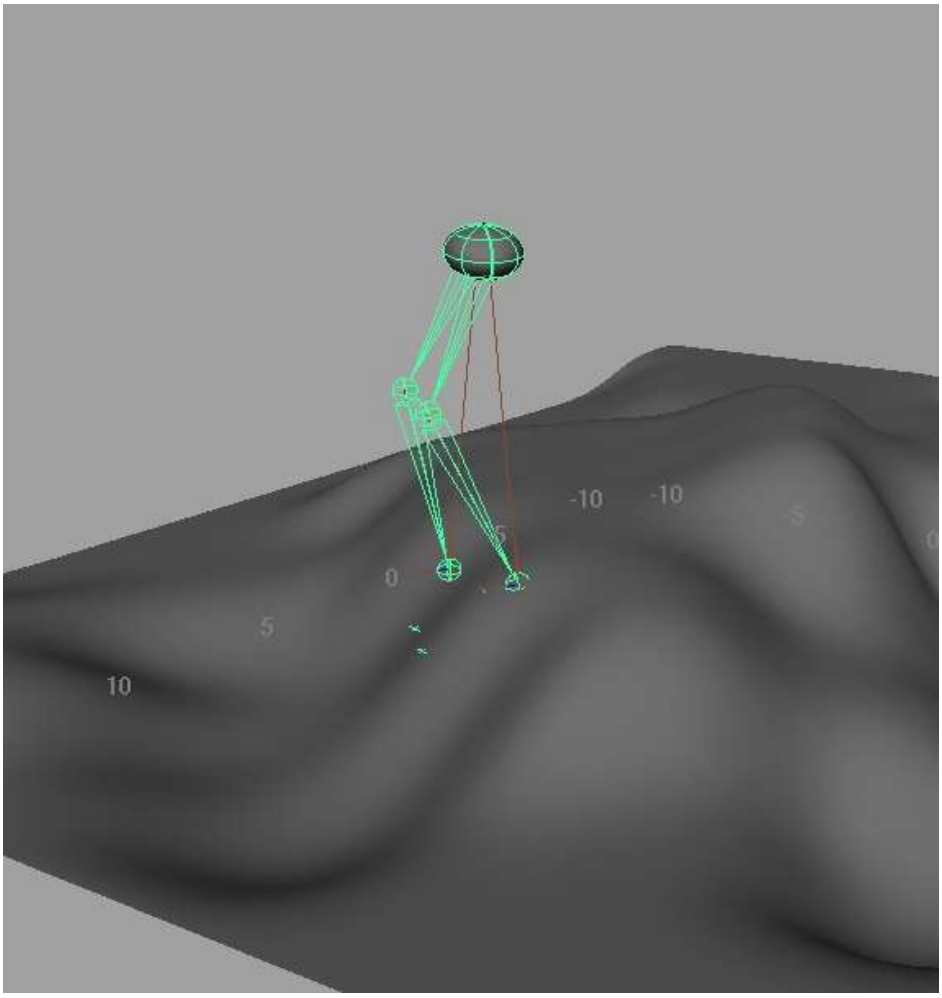
The Automatic Generation of Walk Animation Over Arbitrary Terrain Geometry



MSc Thesis 2005

Siobhan Platten

D1144276

Contents

Abstract

The project objective was to implement a tool to efficiently automate character walk animation over arbitrary geometry. The project succeeded in it's target and successfully generates useful animation sequences in faster than real-time speeds. There are however outstanding stability issues which need to be addressed.

Introduction

A common aspect of animation is the need to move a character from one position in a scene to another. A common method for achieving this is for an animator to define a series of key walk poses which can be looped together to create an 'in place' walking motion. The character is then simply translated across a surface at the appropriate speed to give the impression of walking.

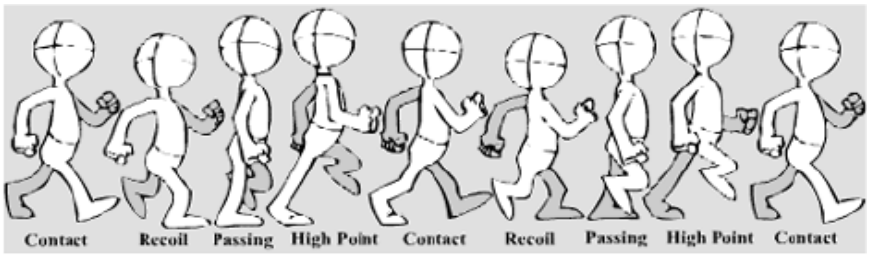


Figure 1: Traditional walk cycle image source http://www.idleworm.com/how/anm/02w/walk1.shtml

This technique is very effective for the common case of a character traversing an even terrain such as a floor. When the character is situated in an environment with uneven terrain however, the animator is faced with the issue of ensuring the character's feet are placed accurately on the surface to avoid penetrating the terrain geometry. The body of the character would also need to be carefully keyframed at each step to ensure the movement accurately accounted for the foot placement. In the case of a manually generated animation sequence, this would require that instead of defining a fixed number of poses to be repeated over the required number of steps, each step would have to be hand animated. If the terrain were highly complex, this would soon become a laborious process and a waste of the talent of a skilled animator.

The aim of this project was to address this issue by developing a tool to automate the process of generating walk animation over uneven terrain surfaces. The particular objective was to develop a solution generic enough to cater for terrain placed in an arbitrary manner in the 3D scene. In order to cater for a wide range of character types, the aim was to provide an extensible solution to cater for any number of feet. It was also decided that the solution should not be Maya specific in order to increase the system's potential for reuse.
The solution was therefore developed as a series of interactions between client and server programs. The server program is generic and can be used by clients developed for any commercially available animation package with the ability for a developer to implement RPC client methods. Alternatively, a proprietary package could be extended to make use of the server. The server's purpose is to analyse the terrain geometry, storing point and normal information in an efficient data structure to represent the terrain's surface. It is the responsibility of the client to provide point and normal information to represent the terrain surface. The client then interrogates the server to find the most appropriate point on the surface for the next foot position. Using this information, the mid-step position is calculated and keyframed ,

and the final foot position is keyframed to complete the step animation. For the purpose of this project, a Maya client was developed. Maya's IK handles are used to position the character's feet, thereby automatically placing the character's knees and hips appropriately

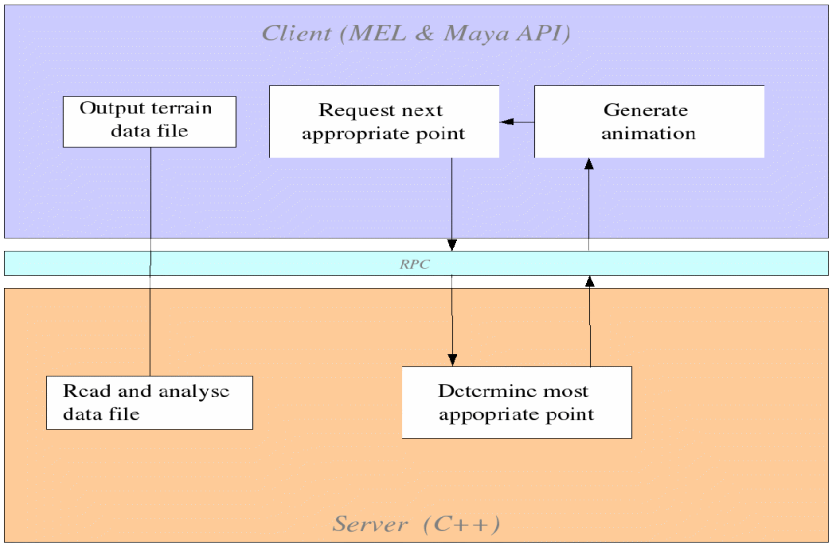


Figure 2: Overview of client-server interaction

Communication between the client and server elements of the implementation is achieved by RPC (Remote Procedure Calling) .

Related Work

Many problems in computer graphics and animation require spatial data to be subdivided. The problem of efficient collision detection has seen a great deal of research. The reason for this is the 'brute force' method of checking each element in a scene for collisions against every other element soon becomes impractical. The computational cost of collision detections increases exponentially as new elements are added to the scene, therefore reducing the number of tests by spatial data representation is advantageous. Solutions for collision detection problems are often hierarchical in nature, such as AABB (Axis Aligned Bounding Box) trees (van den Bergen,2004, p207) . The reason for this is they need to 'drill down' to a point of impact on a model. The problem presented by the need to determine the next appropriate point on a surface is however different in it's nature. Instead of 'drilling down' to a point on a piece of geometry, the problem begins instead at a specified point on the surface and needs to look further afield for a suitable next point. For this reason a non-hierarchical grid based structure as proposed by Hanan Samet (1989) was adopted.

There has also been a significant amount of research carried out to solve the problem of gait generation over uneven terrain. A solution of significant interest proposed by Sun and Metaxas (2001) in their paper *Automating Gait Generation* includes the considerations of step length and step height which also play important roles in this project implementation. The Sun and Metaxas solution however, only accounts for terrain based in the X/Z plane as foot re-targeting is achieved by testing the terrain height at the end of the step. As the author wished particularly to automate the generation of walk animation for insects and arachnids as well as bipedal characters, it was necessary to develop a solution which accounted for arbitrary terrain alignment.

Technical Background

Maya's proprietary scripting language MEL (Maya Embedded Language) was used to fulfill the client side of the solution. MEL provides appropriate commands for extracting the necessary information from a scene file for the terrain data extraction part of the process. MEL scripting is also used for the purpose of generating the step animation at the final stage. MEL does not however provide the means to communicate with a continually running server process external to Maya. To remedy this, a small Maya API plug-in was used to define new MEL commands in order to interrogate the server program via RPC. RPC was developed to provide an intuitive means of developing networked applications. Sun RPC automates the process of sending complex data structures as streaming binary data over a network. The programmer defines the data structures to be used in a C-like language called RPCL (Remote Procedure Call Language). The program `rpcgen` then automatically generates C header files defining the data structures. Client methods to invoke the appropriate procedure in the server process are also automatically generated along with server procedure definitions and code to transmit complex data structures over the network. The RPC standard which defines the network transmission of complex data structures is called XDR (eXternal Data Representation). The reader can find a wealth of detailed information on RPC in John Bloomer's book "Power Programming with RPC" (Bloomer, 1992). Care should be taken however when following the provided examples however as RPC has been updated somewhat since the publication of this book. The reader is advised to consult the many Internet tutorials available to supplement Bloomer's material with up-to-date information.

As efficiency was a stated objective of the tool, C++ was chosen as the programming language to implement the server program. As C++ is compiled and linked prior to execution, it has significant speed benefits over MEL scripting which is interpreted at run-time. C++ also has the advantage of being widely used in the field of Computer Graphics and Animation, this would be an advantage for the ongoing development and maintenance of the tool, were it to be adopted in a production environment.

The server program was developed using some of the principles of object-oriented programming. The requirements of the system were relatively simple so some of the sophisticated elements of object-oriented programming such as inheritance and polymorphism were not required. The definition of logical classes to define entities in the program such as three dimensional points in space grouped together data and methods which affect the data to, providing a more intuitive and robust development environment.

The Solution

The overall aim of the project was to automate the animation of characters over uneven terrain, providing enough flexibility to allow terrain to be aligned in any manner, and multipedal characters to be automatically animated. The aim was not to automate all aspects of the subtle and complex nature of walk animation, but to automate the process of foot targeting which would otherwise be laborious for animators to carry out by hand.

To achieve the overall objective, the project was split into the following subtasks detailed below.

Character setup

As a starting point for each footstep, the system requires the ideal end position for the character's next step. The original solution to this was to create a locator immediately in front of the character to determine it's direction using simple vector arithmetic.

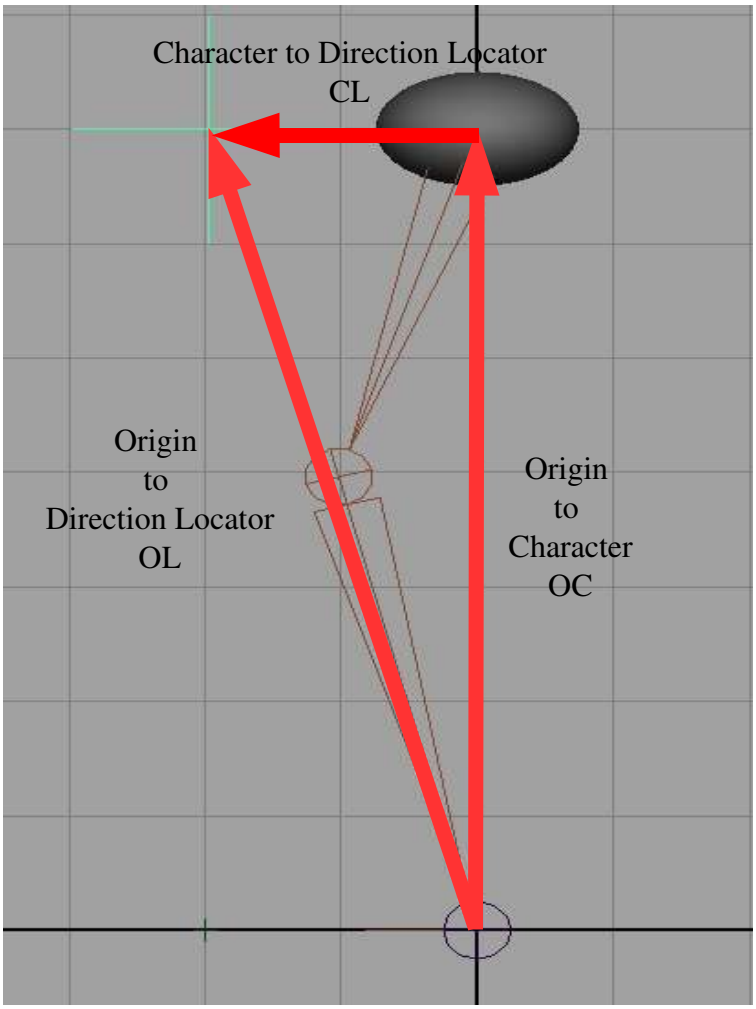


Figure 3: Original solution for determining character direction

As the direction locator was parented to the character, the vector between the character and the locator would always represent the direction in which the character was currently facing. To determine the ideal next position of the character, the direction vector was normalized and multiplied by the user defined step length before being added to the current position of the foot being animated. This solution worked adequately for mildly undulating terrain surfaces, but if the tool generated a shorter or longer step length than usual due to particularly steep terrain, or one foot starting behind the other, this would never be compensated for in subsequent iterations, often leading to one foot lagging behind the other.

To remedy this, a solution which allowed the foot target to constantly be defined as half a step length in front of the character was required. The decision was made to create a locator representing each foot's ideal target. Each foot target locator was created half a step length in front of the foot's starting position, but was parented to the character so as always to represent the ideal next step position.

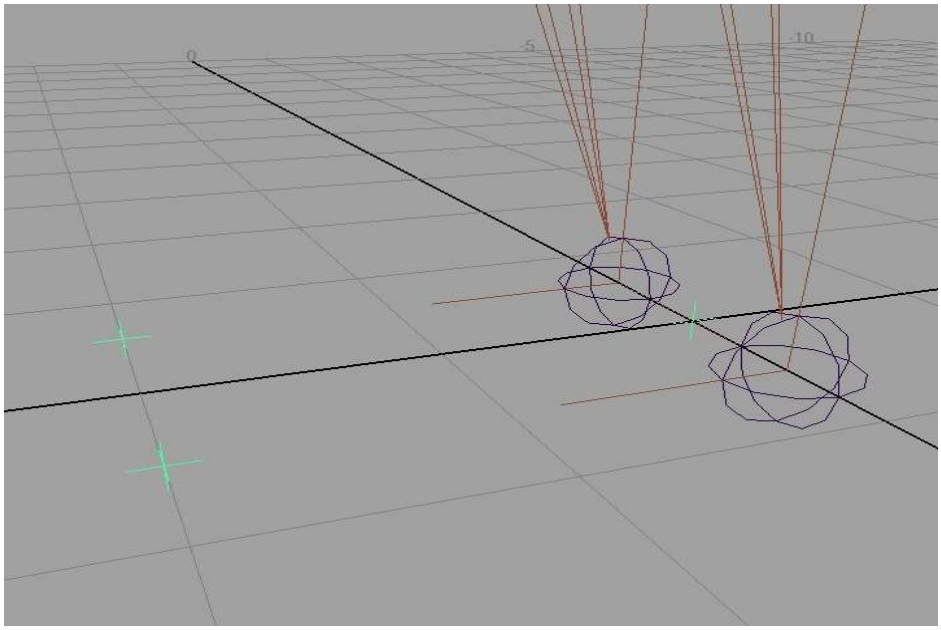


Figure 4: Target locators for each foot

This approach resolved the foot lagging problem.

The next problem encountered with the determination of the ideal next position was that the foot target locator solution tented to generate overly short steps on steep terrain as the target locator would still be parallel to the XZ plane. This was addressed with the addition of a third locator between each foot pair. The foot target locators were parented to this third central locator which was in turn parented to the character. The central locator was then constrained to the terrain normal to help maintain the appropriate step length and direction in steep terrain scenarios.

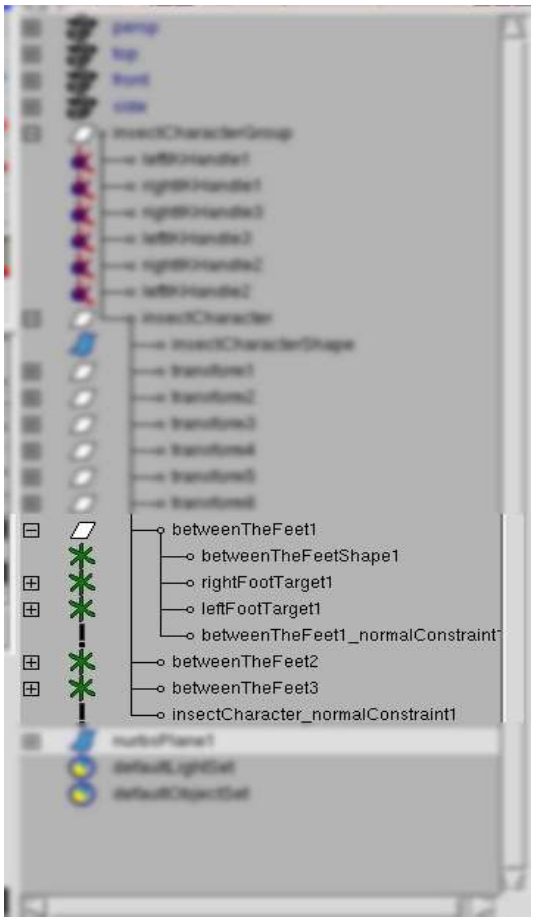


Figure 5: Normal constraints help maintain appropriate foot targets.

In the case of multiped characters such as insects and arachnids, it was found to be useful to also constrain the character's body to the terrain normal. Bipeds tend to attempt to remain upright so this was unnecessary in the case of these characters.

The creation of these locators by an animator would be tedious and prone to error, so the tool was developed to automate this process as a preparation step prior to generating animation.

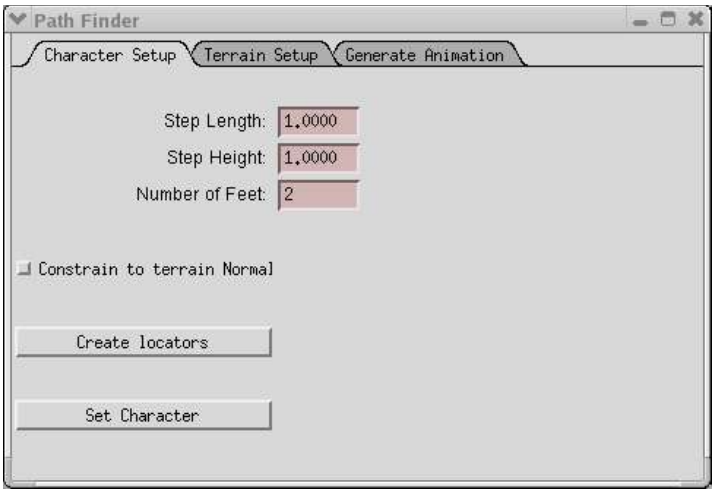


Figure 6: Character Setup panel

It is a prerequisite that the character be pointing directly in the positive Z axis in order for the locators to be created accurately, otherwise the procedure is entirely automatic.

Terrain setup

Data extraction

In order for the server program to function, it needs to be provided with point and normal data describing the terrain' ssurface. To achieve this, the tool was developed to include a second pre-animation preparation stage.

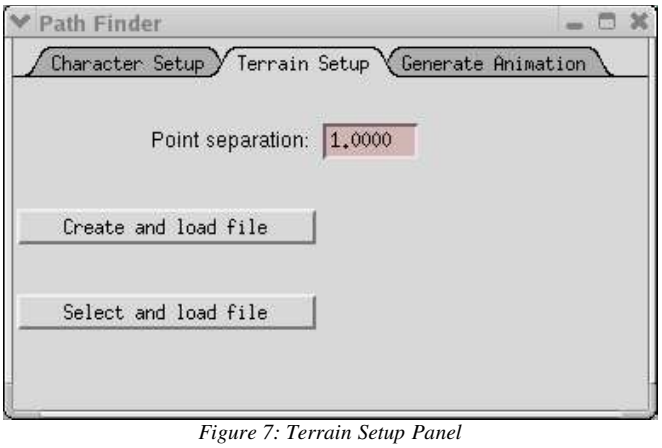


Figure 7: Terrain Setup Panel

The panel provides the option to create a data file describing the currently selected geometry, or load a previously created file. This option provides the ability to have a character walk over terrain geometry which is not actually present in the scene. This could be useful if the terrain data were extensive as the geometry would not need to be present in the Maya scene, thereby speeding up viewport response. To cater for this requirement, a MEL script was developed to analyse the geometry and output an ASCII file containing the point and normal information required by the server program. The tool was developed to deal with solely with terrain defined by NURBS geometry. Maya supplies several useful commands and nodes for extracting information from NURBS geometry.

The script creates a `pointOnSurfaceInfo` node which is connected to the `worldSpace` attribute of the terrain geometry. This allows world space coordinates of points on the geometry's surface to be accessed by supplying the desired position in the object' sUV space. It is an important aspect of the tool to be able to specify the number of points generated to represent the terrain geometry's surface. If too many points are generated, there will be greater computational overhead when the server program comes to select the next appropriate point. On the other hand, if not enough points are generated, there will not be enough information to accurately represent the terrain surface. The tool therefore provides a user defined ' PointSeparation' input to the data extraction step. This allows the user to decide on the balance between efficiency and accuracy based on the requirements of the sequence to be generated.

Once the desired point separation has been input, the script relates the required step length in world space units to the equivalent in the object's UV space. This is achieved by the use of an `arcLengthDimension` node which provides the required length in world space units along a curve on the object' ssurface.

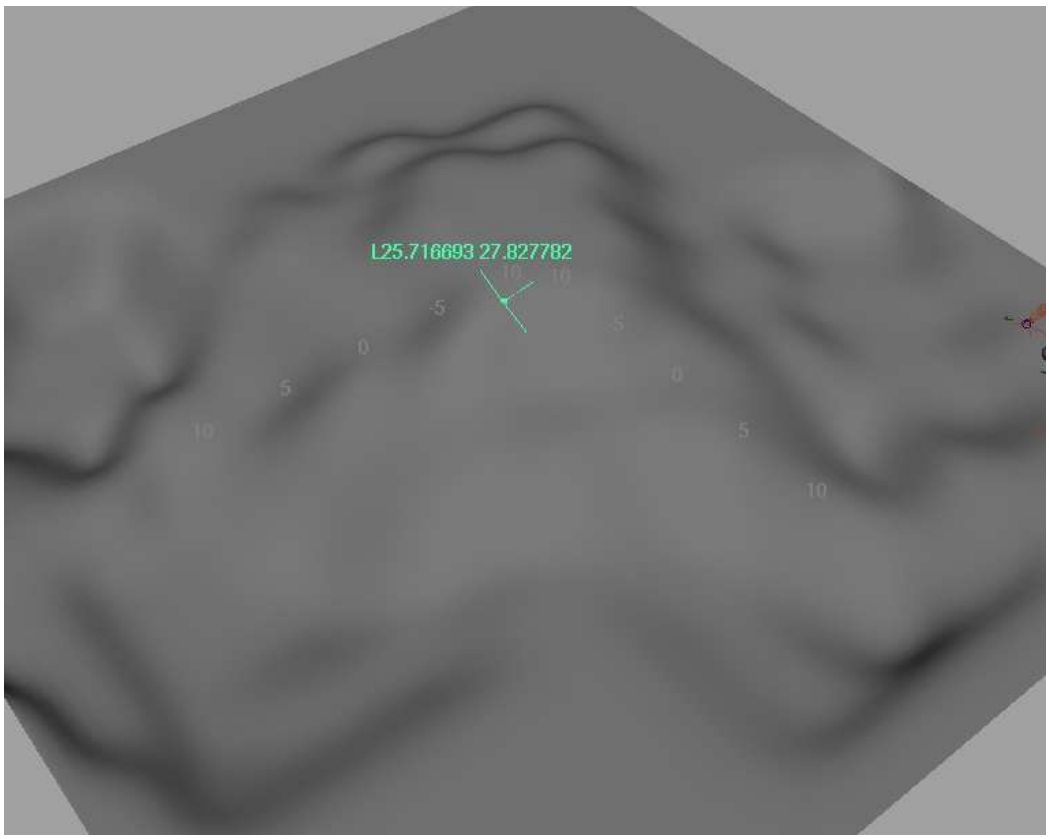


Figure 8: `arcLengthDimension` *node relates UV space to world space*

The length of the curve is then divided by the desired point separation value to give us the appropriate interval distance in UV space. A double loop in the script then samples the surface at this interval along the U and V axes, extracting the world space coordinates and normal information from the `pointOnSurfaceInfo` node at each iteration.
The point and normal information is then written to the ASCII text file ready to be parsed by the server program in the next phase of the animation generation.

Data Analysis

The server program parses the text file produced in the previous step and uses the information to create a logical data structure representing the terrain surface.

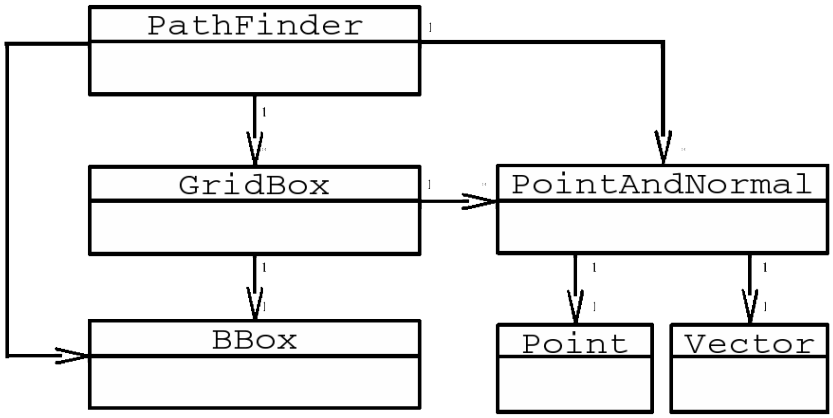


Figure 9: Class diagram of the server program

Each line of text in the ASCII file being parsed represents the x y and z coordinates of a point on the terrain surface, and the xyz coordinates of a vector representing the surface normal at that point. As each line is read, a `PointAndNormal` object is instantiated using the parsed values. The `PointAndNormal` object is stored in a vector data member of the `PathFinder` object.

Having stored the representation of the terrain surface as a set of points and normals, the data then needs to be subdivided so it can be accessed efficiently by other methods. As the search for the next appropriate position on a surface begins with a finite point and expands outwards, a non-hierarchical fixed-grid data structure was judged to be appropriate.

> "If range queries are made using only a fixed search radius, the fixed grid is an efficient representation. It is also efficient when the data points are known priori to be uniformly distributed over space"
>
> (Samet,1989, p47)

To create the grid data structure, the program first creates an Axis Aligned Bounding Box (AABB) to encompass the entire terrain geometry. This is accomplished by determining the minimum and maximum x y and z values of the surface points. These values are then used to define the bounds of the AABB. This AABB is then subdivided based on the proposed step length of the character to be animated. It is important to ensure that each point within reach of the character's current foot position is contained in a neighbouring grid cell. For this reason the grid boxes are defined as being slightly bigger than the specified step length in world space units.

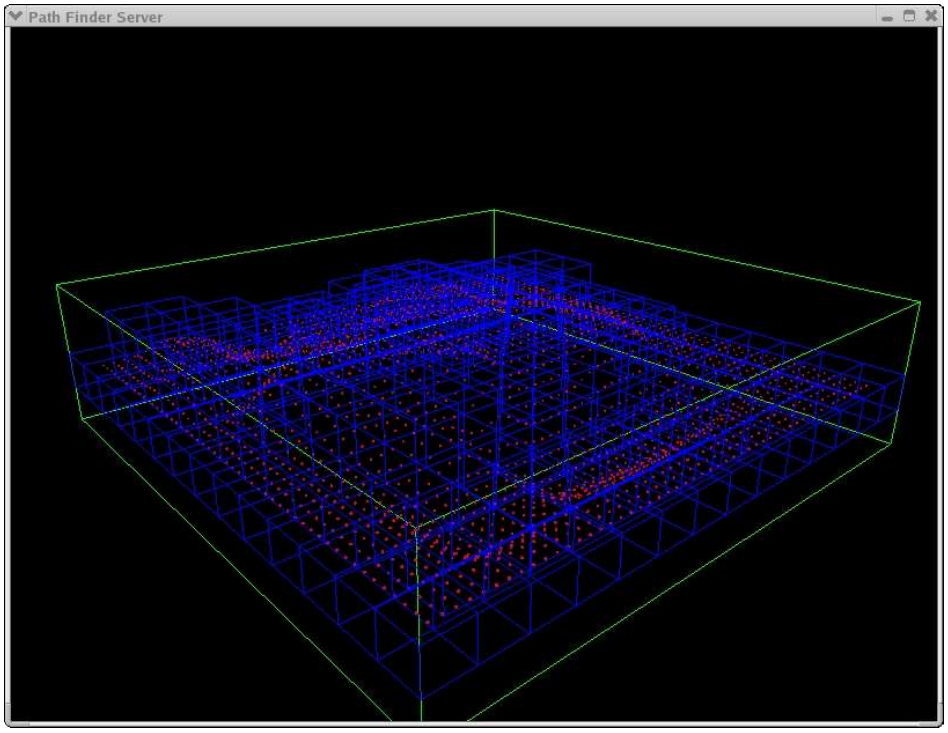


Figure 10: The terrain data is subdivided into a fixed-grid, non-hierarchical data structure

Each grid box is identified both by it's position along the x, y and z axes and by its one dimensional array index. In order to identify the spatial neighbours of a given grid box, the program adds or subtracts the x y or z value of the source grid box as appropriate. As the program stores the grid boxes in a one dimensional array however, methods were required to switch between the two means of grid box identification. In the case where the array index is known and the XYZ indices are required, the index is divided by the number of grid boxes contained in a single layer in the YZ plane. This gives the X index of the grid box. The remainder of this division operation is in turn divided by the number of grid boxes in s single row along the Z axis. This gives the Y index. The remainder of this division operation gives the Z index. In the case where the one-dimensional array index is known and the XYZ indices are required, the reverse operations are carried out. The X index value is multiplied by the number of grid boxes in the YZ plane, the Y index is multiplied by the number of grid boxes along the Z axis and finally the Z index is added to the result to give the array index.

It was also necessary to identify the appropriate grid box index given a specified point in three dimensional space. An example of this is the 'Snap Character To Terrain' facility provided by the tool prior to the generation of animation. The animator positions the character's feet near to the terrain geometry and the server program works out the nearest point located on the surface. The appropriate grid box from which to start the search is determined by calculating the percentage of the point's position along each of the three coordinate axes in the main AABB encompassing the terrain geometry. This percentage value, multiplied by the number of grid boxes along each respective axis gives the appropriate XY or Z index value.

Animation Generation

The generation of the walking motion is achieved by a close collaboration between the client and server elements of the tool. The overall control of the process lies with the MEL script. The GUI allows the

user to define the number of steps to be generated in the sequence and the number of frames the animation should take place over. The start frame for the sequence is also defined by user input. The number of steps specified gives the control value for a loop to generate alternating steps. Steps are generated two at a time so an even number of steps will always result even if the user specifies an odd step number.
To generate each step, the character's feet and body positions are first keyframed before any motion takes place. The current positions of the feet to be moved are then determined using MEL's `xform` command in query mode. To determine the next appropriate foot position on the terrain surface, the server program also requires the direction the feet are to be moved in, and the length of the step to be generated. This information is extracted by using simple vector mathematics to determine the vector between the foot's current position and it's associated target locator. The vector itself defines the direction of the required point, and it's magnitude gives the target step length.

The server program then takes this information to calculate the 'ideal' next point in a straight line along the direction vector from the foot starting position. The program then compiles an array of candidate positions for the next foot position. This array is comprised of the `PointAndNormal` members contained in the source point's grid box, along with those occupying the neigbouring grid boxes in the direction the character is facing.

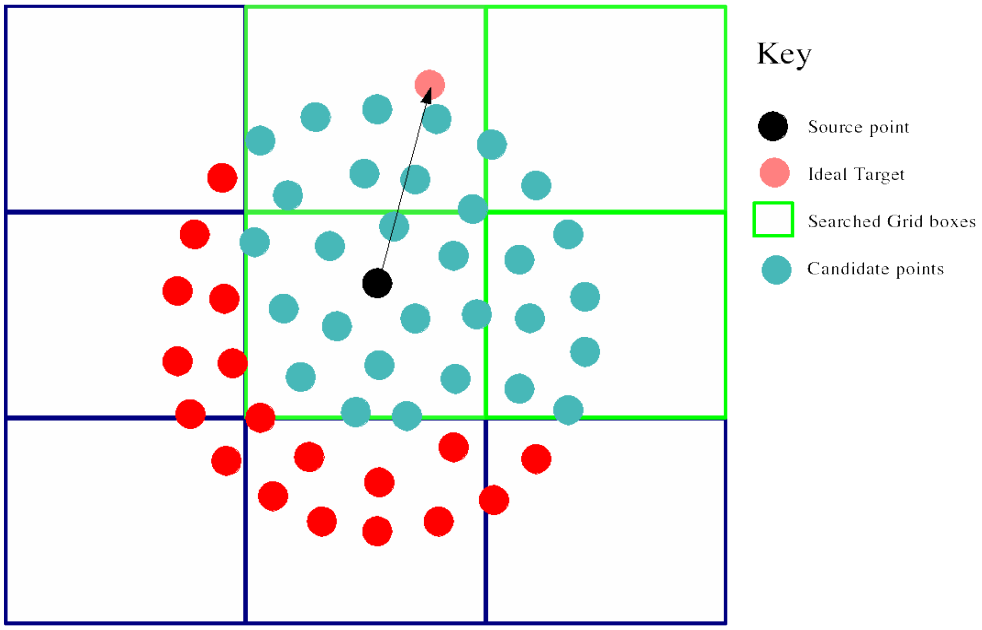


Figure 11: 2D illustration of the target point searching technique

The appropriate grid boxes to be searched are determined by analysing the direction vector parameter. For example, if the z component of the vector is negative, only grid boxes below the source grid box will be searched.
When the list of candidate points has been compiled, the candidate whose vector to the 'ideal' position has the smallest magnitude is selected and it's position is returned via RPC to the Maya client. This searching technique has the advantage that only a finite number of candidate points will be checked for

suitability regardless of the size of the terrain geometry.
Given the location of the foot's start and end position on the terrain, the MEL script then calculates a suitable mid point position to keyframe. The end position is also keyframed to generate an arc motion for the step.

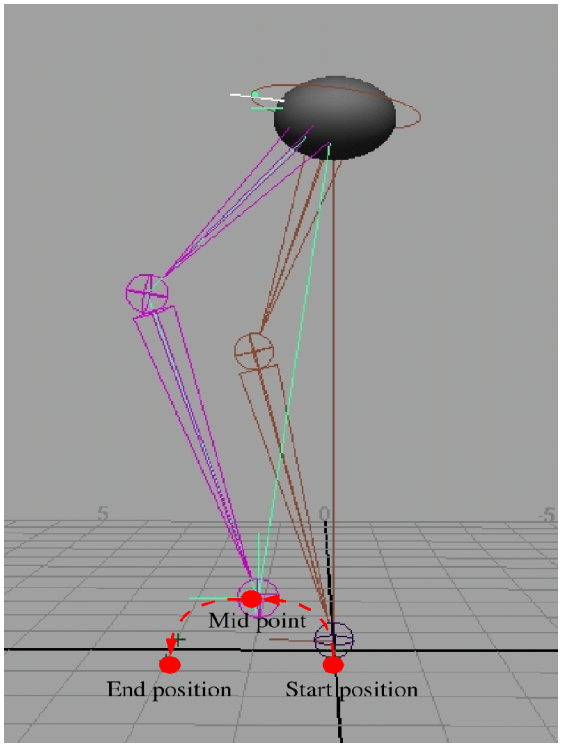


Figure 12: Start, mid and end points define the step animation

In order to place the midpoint correctly, the MEL script determines the point halfway between the start and end position by calculating the vector between the two points, halving it and adding the result to the start position. This gives the appropriate distance from the starting position for the midpoint, but the point needs to be raised above the terrain surface. As the tool is designed to cater for terrain orientated in any direction, the direction in which to raise the midpoint needs to be calculated. The walk generation script caters for this requirement by interrogating the server program to obtain the normals associated with the start and end positions of the step. These normals are then averaged, and multiplied by the user specified step height to give the correct position for the midpoint.
The character is rigged to have an IK handle on each foot. The IK handle is keyframed at the start, mid and end positions of the step. This approach automatically gives the correct rotations and positions for the knee and hip joints. The final stage is to add half the vector between the foot start and end position to the character's body position and keyframe accordingly. The script specifically keyframes the translation attributes only of the character's body. This is so as not to override the character's rotations which will have been automatically set if the character is constrained to the terrain normal. In the case of multiped characters, the vector of all the footsteps is averaged and halved before being added to the character's position.

Development strategy

As the C++ server program was largely concerned with three dimensional data, it was useful during the development process to be able to see a visual representation of that data. To achieve this, the program was developed with the ability to operate either in 'server' or 'visual' mode. In visual mode, the program takes the filename, start position, direction vector and step length as command line parameters and uses

OpenGL to generate a snapshot view of the data. This technique was particularly useful when developing the methods to relate XYZ indices to the one dimensional array index of the grid box elements as the solution to this issue was easier to work out with a visual aid. Unfortunately the server program could not run in server and visual mode simultaneously so the parameters to the visual mode needed to be chosen carefully to reflect the scenario of interest.

Known Issues

When the tool has been used to generate several sequences, the Maya client occasionally loses contact with the server program. The end result causes a segmentation fault within Maya. The fault is intermittent, and it is unclear at this stage whether the problem lies in the client, server or RPC part of the implementation. Careful investigation is required in this area as important data could potentially be lost were the tool to be used in a production environment.

The submitted version of code failed to correctly identify neighbouring grid boxes in the positive Z direction. The result of this was that the animation sequence would prematurely halt as the server program failed to locate a suitable next position when the character was travelling in the positive Z direction. This issue has since been addressed.

The original intention of the system was to allow the user to specify changes of direction during the animation sequence. Unfortunately the difficulty of controlling the character' spivot point during keyframing has resulted in this feature not being implemented in the project timescale. It was found however that the use of foot target locators allowed the animator to record a specific number of steps, manually centre the character's pivot point and then merely keyframing the character's rotation. A second automated animation sequence can then be generated from a specified frame. This process is not overly laborious and in fact allows the animator a greater degree of control over the change of direction.

Conclusion

Suggestions for Development

Polygonal terrain

The implementation as it stands only caters for NURBS terrain geometry. The usefulness of the tool would be improved considerably if polygon meshes could also be used to define the terrain. Several issues would need to be addressed for this to be implemented. Firstly, the extraction of point data representing the terrain surface would need to be addressed. A possible candidate solution would be to query the world space coordinates of each of the polygon vertices. This solution would however have several drawbacks. The number of vertices in the terrain model would not be under the control of the tool, therefore it would not be optimised based on the step length of the character to be animated. There would be a risk therefore that vertices would be placed too far apart for the character to 'reach' to the next one.

Conversely, there also be a risk of too many vertices defining the terrain thereby affecting the efficiency of the next point algorithm. To negate these risks, the data extraction phase of the tool would need to be augmented to disregard extraneous point information, and extrapolate additional points as required. In addition to this, Maya' sautomatic normal constraint facilities are only available for NURBS geometry. If the solution were to be developed for polygon terrain geometry, calculations to implement these

normal constraints would need to be developed. The surface normal at each polygon vertex is available for interrogation. The normal at each foot position would need to be collated and averaged and the 'up' vector of the character would then need to be amended to match this average normal. This up vector could be defined by adding an additional locator directly above the character in the character setup facility of the tool.

More efficient use of Inter-Process-Communication (IPC)
The use of RPC in the project gave significant benefits. It is important to bear in mind however that the use of a client-server solution introduces a potential network communications overhead and RPC methods should therefore be used judiciously. The current implementation uses separate procedure calls to interrogate the server for point and normal information during the step generation loop. These data items could be retrieved in one step, thereby reducing the message passing overhead.

Extra realism
The current implementation successfully solves the problem of character foot placement over uneven terrain. The placing of feet is however only a small part of the implementation of a convincing walk animation. The implementation of a character's walk is widely accepted to be one of the most important features, in a visit to the NCCA in October 2004, Mike Milne of Framestore CFC stated that Framestore's best animators are used to define walk cycles as they are so important for characterisation. In his book "The Animator's Survival Kit"(Williams, P102-211), Richard Williams goes into extensive detail on this subject, further confirming the subject's importance. In this context, an automated solution would most likely not be able to cater for the subtleties of emotion conveyed in a character's walk. The aim would rather be to convey the physical nature of the movement of hips, shoulders and arms as affected by the character's terrain. These features could be included in the scope of this tool. For example, steep terrain would have an effect on the step length of the character as well as the positioning of the midpoint of the step, the gradient of the vector between the start and end foot position would provide the tool with enough information to generate an automated solution. The ultimate aim of the tool would be to remove the requirement for a skilled animator to define the tedious elements of walk animation, leaving them to concentrate on the elements which contribute to the aesthetic quality of the sequence, thereby using their skills to the best advantage.

Evaluation
The implementation was successful in meeting the stated objective of automating character walk animation over arbitrary terrain geometry. The solution was also found to be efficient – generating walk sequences faster than real-time. The use of RPC to enable the communication between Maya and external processes has relevance beyond the scope of this project. In the case of this project, the data server was located on the same computer as the Maya client. This does not necessarily have to be the case. If a multi threaded solution were possible, commands could potentially be distributed to any number of computers over a network. This could prove to be extremely useful in cases where intensive calculations are required as a facility could tap into hitherto unused processing power.

References and Bibliography

1. BENTLEY, J L. 1980. *Multidimensional Divide-and-Conquer.* Communications of the ACM Volume 23 Issue 4. April 1980, New York
2. BLOOMER, J. 1992. *Power Programming with RPC.* Sebastopol, CA. O'Reilly.
3. SAMET, H. 1989. *The Design and Analysis of Spatial Data Structures.* Addison Wesley
4. SUN, H.C. & METAXAS, D. 2001. *Automating Gait Generation,* Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques. New York.
5. VAN DEN BERGEN, G. 2004. *Collision Detection in Interactive 3D Environments.* San Fransisco. Morgan Kauffman
6. WILLIAMS, R. 2001. *The Animators Survival Kit*, London, Faber and Faber