Available online www.ejaet.com

European Journal of Advances in Engineering and Technology, 2015, 2(5): 18-22

Research Article ISSN: 2394 - 658X

Dynamic Terrain and Character Generator Tool for Game Engines

Martin Bastian

Department of Information Technology, Nizwa College of Technology, Nizwa, Oman
martin.bastian@nct.edu.om

ABSTRACT

Dynamic Terrain Generator creates terrain and character elements interact with the terrain dynamically during the start and duration of the game .Dynamic terrain tool allow player to navigate unlimitedly through all four directions with newly generating terrain fields as progress forward without consuming system resources too much. It gives the illusion of a real world that gives user the feeling to move anywhere he want while playing the game. It is suited for war games, hunting games and modified version can also use for racing games

Key words: Virtual reality, game design, game engines, memory management, interpolation algorithms

---

INTRODUCTION

Development of dynamically generating terrain tool requires a study about terrain layouts. Analysing any terrains or landscapes we can easily notice some kind of uniformity or similarities in that particular terrain. We can see the each terrain is distinguished from each other by the difference in the elements like trees, bushes, terrain structure etc. Analysing the element like tree in a terrain we can actually count almost all the tree types in that landscape and more over most of the terrains have some signature trees ,bushes or other elements that distributed widely in that terrain. For example tropical beaches have coconut trees widely distributed and alpine trees in high altitudes. This selection of elements helps to identify or minimize the number of elements needed to generate dynamically in a terrain.

Background artists create terrains or landscapes for games and Animations using a systematic approach. First they will get the 2d layout design from the layout department along with the storyboard. Background artists now prepare the backgrounds using their imagination and references from real world that support the story sequence. First they will plan and create the farthest layer like sky. After creating the first layer they will add the second layer like far mountain range or city sky line. Then they will add third layer like hills or trees that displayed much nearer after the sky and mountain range. At the end they will add the nearest elements that usually interact with character or props.

LAYERED APPROACH

Dynamically generating background requires these layered approach .First we analyse the farthest layer like sky. In 3D design software's BG artists create a big sphere or semi sphere and map it with large size image of sky. Sphere or semi sphere will cover the entire terrain so it gives 360 degree coverage of sky. So in any camera angle sky will be visible. Mapping the sky to the sphere using proper method and the size of sphere gives the reality needed.

Sphere size shouldn't be too large as dynamically generating terrain also removes unwanted terrains. The generation of dynamic terrain and characters also depends on the game environment. Let's take a case study of a war game. The game consists of terrain that is populated with trees and bushes. Player has the role of a soldier and moving through the terrain in eye view so that the camera view itself is the movement of the player. As the player moves along the terrain, dynamically generated enemy soldiers will attack the player.

Second Layer

The next farthest layer like mountain range is usually faded and has a silhouette visual with some common tone. This layer should automatically generate with a combination of five or six different mountains that selected randomly to form the mountain range. Let say five mountains created and stored in the library the possible combinations for five mountains is 5 factorial that is 120 and also with different sizes and flipped horizontal shapes. So every time the game starts a combination will randomly generate and placed in front of camera as second layer

after the sky layer. Whenever the camera moves another different combination will randomly generated and attach with the extended end (in the direction of camera pan) of the first combination .So it always keep the asymmetry needed for the backgrounds.

Fig. 1 Crowded coconut trees in tropical place [3]

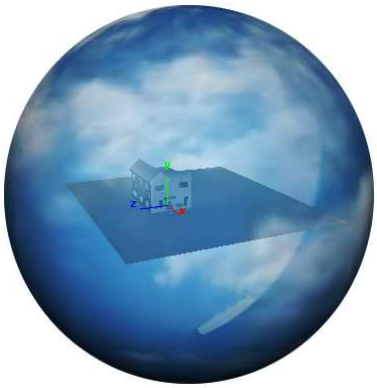

Fig. 2 Sky wrapping the entire terrain using the sphere primitive [3]

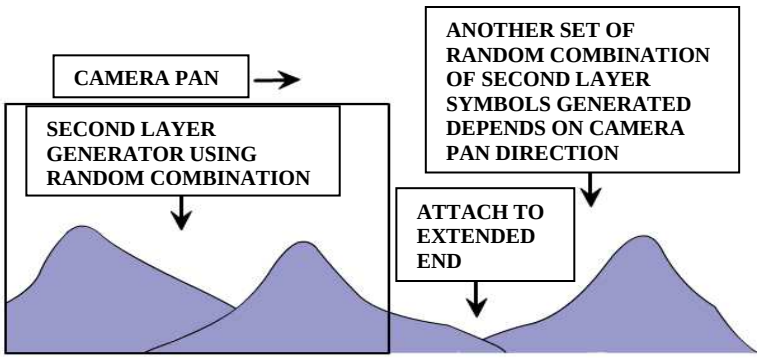

Fig. 3 Attach to the extended end of first combination [3]

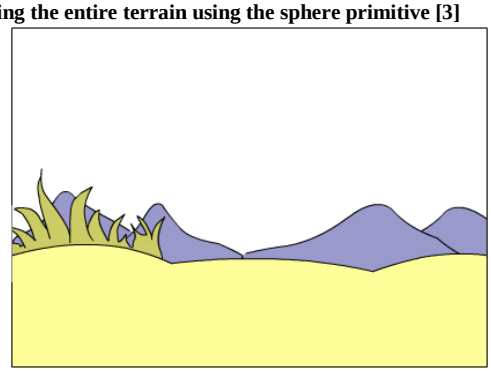

Fig. 4 Partially hided mountain range by border hills [3]

Third Layer

Next layer elements will be more distributed in Z axis or depth axis .First we have to generate the plain which placed in the 3d space with center of plain at x,y,z (0,0,0).Creating the feeling of depth like real world is tricky task for graphic artists. Graphic artists filled the plane with elements like trees or buildings according to layout design. The border of plane is populated with small hills or bushes that give a partial view of far away mountain range or second layer. Even though the second layer mountain range is placed nearer to much closer third layer the texture and color of second layer makes it looks very far and the partially covered hills give the illusion of depth and the original distance between two layers is hided. Fig. 4 depicts the partially hided mountain range by border hills of plane to give the depth illusion and it also helps to add new plane after the border of the current plane successfully hiding from the user.

Game starts with the placement of a plane from the library of predesigned set of planes. As the game progresses the user or player moves forward and reach the end of plane in one of four directions. Once player reaches the end of the plane world coordinate reading shows user reached the border of plane and another plane will add to the extended end which also has borders with hills and bushes that partially cover the second layer far mountains. Now the previous plane can be removed if not comes in view and the second layer (mountains) will shifted behind the border of new plane if needed. Shifting is done usually during the player movement from first plane to second plane where the camera view is cheated by showing sky (like the eye view of a player jump from hill to ground) to hide shifting of planes from the player. The number of planes needed is very less at least four as previous planes can be added again because the elements like tree in a plane are dynamically generating which explained below.

Dynamic Generation of Elements

In real world planes are very much asymmetry and the procedural algorithms of 3D software that generate bump maps for the plane surfaces gives the texture needed for the plane to look real and dynamic. A procedural algorithm doesn't add any extra vertices and will not change the topology of plane. Every plane is made of vertices. After the placement of plane store the coordinate information of each vertex of that plane using a code like that given below which is using in Maya.

```
#include<maya/MFnMesh.h>
Void outputMeshVertices(MObject &obj)
{
//attach the function set to the object
```

```
MFnMesh fn(obj);
//this will hold the returned vertex positions
MPointArray vts;
//use the function set to get the points
fn.getPoints(vts);
//write number of verts
cout << ” numverts ” << vts.length() << endl;
//only want non history items
For(int i=0;i!=vts.length();++i)
{
cout <<vts[i].x<<” “
      <<vts[i].y<<” “
      <<vts[i].z<<”\n”
}
}
```

After getting the x, y and z co-ordinate information of each vertices the random function selects some vertices to place the element like signature tree in x,y and z direction. Another group of vertices will select to place secondary elements like bushes, rocks etc.

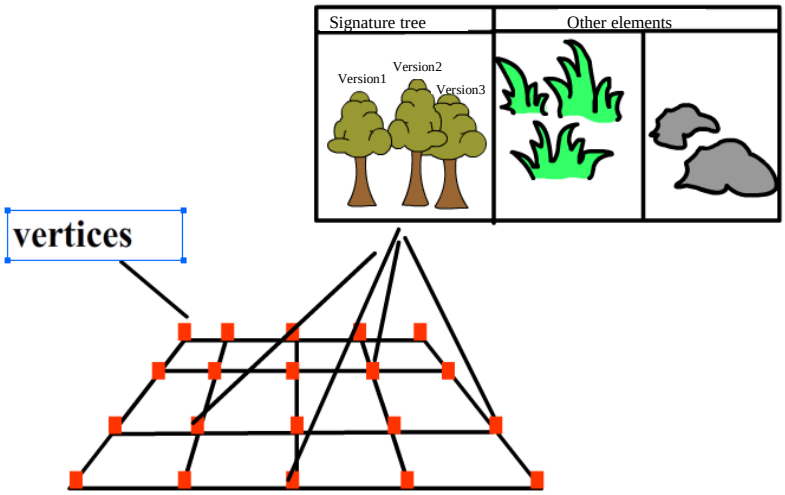

Fig. 5 Selecting random vertices to place elements in Terrain [3]

Pivot point of elements can be used to place elements randomly. Randomly selected vertices x, y and z coordinates will be the value of elements pivots.

```
Object1.curentpositionx = Vt[1].x
Object1.curentpositiony = Vt[1].y
Object1.curentpositionz = Vt[1].z
```

After the generation of terrain and its elements the camera movement is possible in any direction and virtually no limit for the movement to any direction. The dynamically generating procedural bumps of the plane and the random placement of elements make sure the asymmetry needed to give the reality even though the planes are reusing again and again in the duration of game.

Dynamically Generating Characters

After the terrain generation the next major task is the dynamic generation of game characters. Every character in the game will have a predefined set of movements. For a war game each enemy soldier will have some predefined movements. Let’s say each enemy will have a defensive run cycle, shooting action, hiding action and peeping action. Enemy soldiers can be come under one common class named enemy with these four predefined movements as four methods of this class as enemy.run(), enemy.shoot(), enemy.hide() and enemy.peep(). From this class any number of enemy instances or enemy objects can be generated like enemy1.run(),enemy2.run() etc. The generation of enemy in the duration of game requires staging of characters and the movement of character across the terrain. Staging or placement will have to done only in the camera view at any time in the duration of game. To avoid the sudden appearance of characters which affects the reality it is always better to place the characters behind the terrain elements like tree or building and comes out or appear during their movement. The random selection of terrain elements to place the character gives the dynamicity and reality needed for the placement of character.

Selecting the terrain elements needed for the placement of character again depends upon the camera view. After the game starts camera view coordinates will always keep track of the elements that is currently in camera view. If the

pivot coordinate of an element is inside camera view then random function may or may not select that element from the set of elements in camera view to place the character.

Each character has a set of predefined movement like methods of an object and calls that movement randomly. Animators for game development create the animation cycles or actions needed for a character in a game. Most of the cycle actions like run is creating using classical animation style where all the poses of a run cycle was created without having a forward movement so that it will look like a treadmill run.

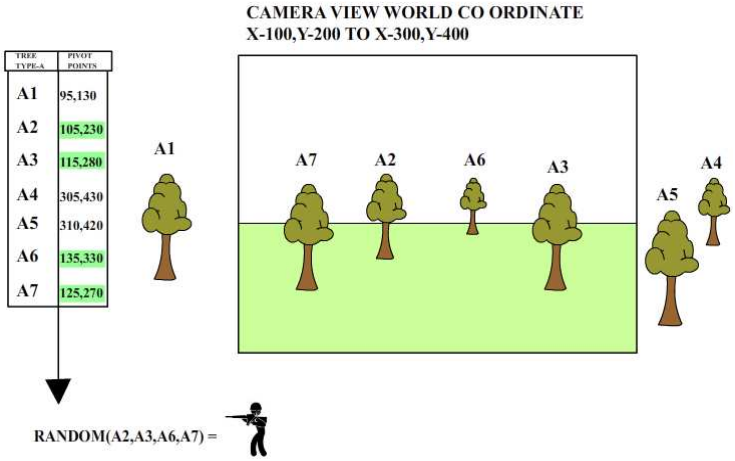

Fig. 6 Generating character movement

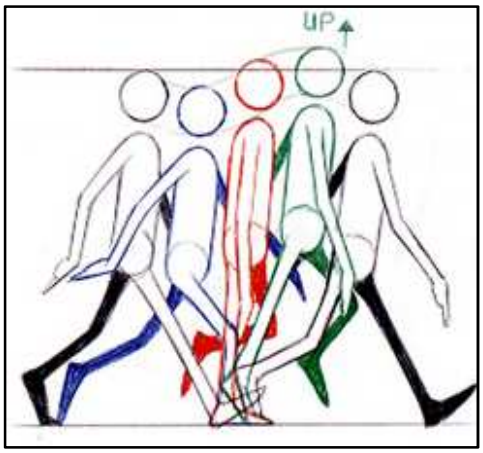

Fig. 7 A typical walk cycle [4]

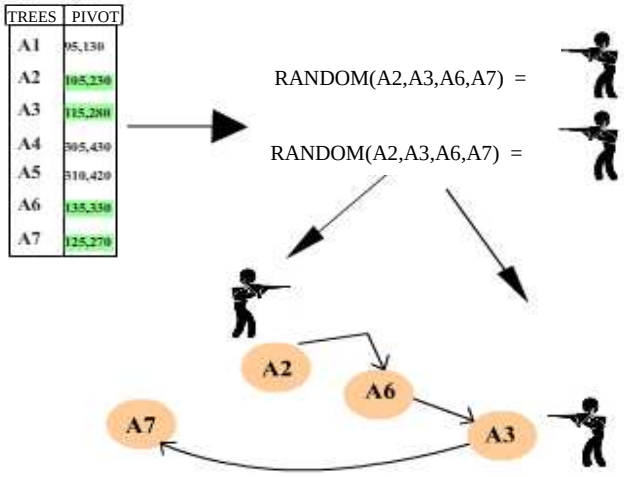

Fig. 8 Placement and generation of character's path

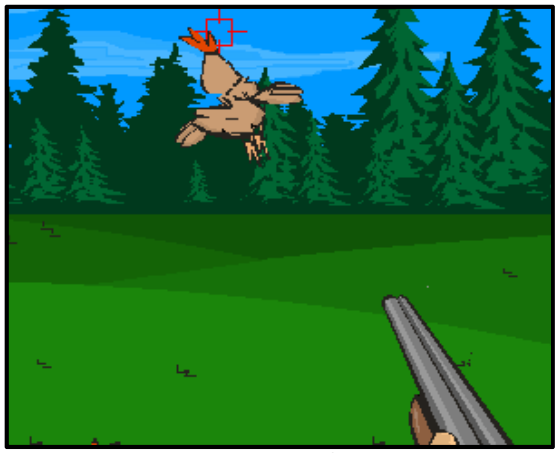

Fig. 9 Camera View with the tip of the gun as player's eye view [5]

These animation cycles help to execute the forward movements separately whenever needed randomly and dynamically. So to finish a run action character has to call two methods first the run cycle and it makes character to run on the same spot without having a forward movement like a treadmill run. This run cycle happens during the random placement of character behind the terrain elements and the player cannot see this characters which is on the spot run cycle. When a particular character is called for action then the second method of forward movement will call with start and end coordinates. Example is given below -

Place
placed behind A2 Tree - enemy1.runcycle()
placed behind A6 Tree - enemy2.runcycle()
Call forward movement
Enemy1.move(x1,y1,z1,x2,y2,z2) –character start from x1,y1,z1 to x2,y2,z2 co-ordinate
Enemy1.move(x2,y2,z2,x3,y3,z3) –character start from x2,y2,z2 to x3,y3,z3 co-ordinate

All the start and end co-ordinates will be terrain elements like signature tree for enemy characters to hide like shown in Fig. 8. The selection frequency of terrain elements to place characters in the camera view at a particular duration of time can also be done randomly with a minimum time interval between each selection. The selected elements pivot coordinates will be used to place and generate the start and end points for the character movement. All other actions like shooting, hiding etc. can also generate randomly. The characters are always placed and move in relation with the current camera view at any particular time of the game. So the player can move 360^0 direction to any distance like a real world and still can find and fight with enemy soldiers. Unlike usual games that the player is

always going fixed path the dynamic generation of terrain and characters give the real life experience of unpredictability and exploration. Player movement is showing like eye view so the movement of camera in any direction itself is the imitation of player eye view.

Collsion Detection of Characters

Character movement along the terrain requiers to avoid collision with the terrain elements like tree ,bush etc. After the generation of character path the next step is to find the terrain elements along the path.Use collision detection algorithms like below.

Let the start position of character path is (x1,y1,z1) with values (100,200,300) and end position is (x2,y2,z2) with values (300,400,500) .Now we know any element in x value between 100 to 300 , y from 200 to 400 and z value between 300 to 500 is in collision with character movement in the path.Also we have to consider the total volume of the element in x,y and z direction.Now this requiers the data that generated when game artists created this objects or elements.This data comprises of height, width and depth of the element. So any character move in the path will collide if it intersects the element's x + width or y + height or z+ depth.

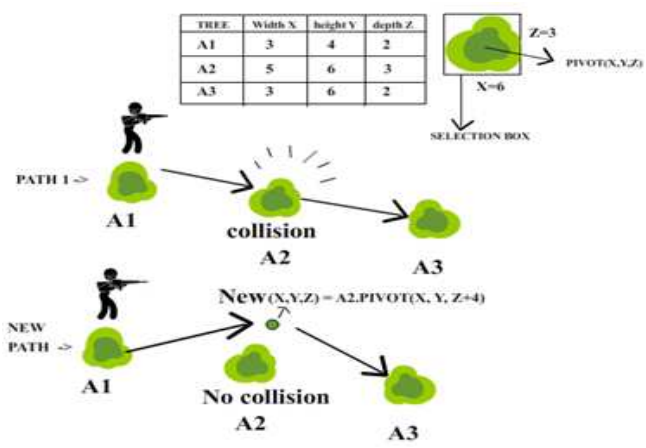

Fig.10 Collision of character path

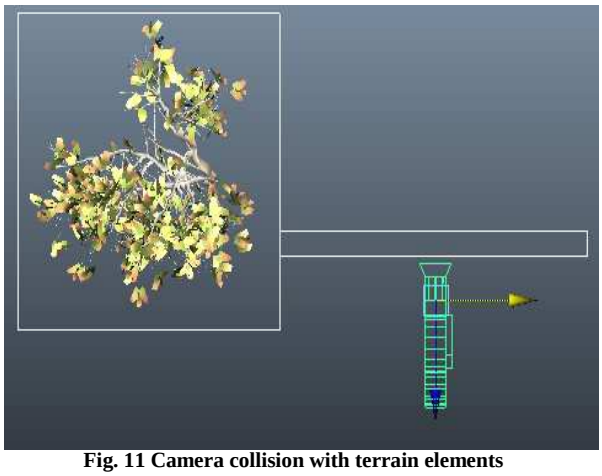

Fig. 11 Camera collision with terrain elements

Many advanced functions are also available according to the software using to detect the collision in game programming to ease the task.Player movement can use nearcliping property of camera to avoid moving inside through the mesh or collision detection of camera is good to implement to increase the reality of the game. If the edge coordinates of camera intersect the bounding box of terrain elements it restrict the movement of camera as shown in Fig. 11.

When the character moves along the terrain using the generated path it is important that characters foot touching the ground irresepective of terrain's dynamically generated irregular shape of slopes and hills. One possible solution is to store all the vertex co-ordinates of the generated path so that the root of character touches all the vertices as move along the path. If the distance between two vertices of the path is too much there is a possibility that the character root will not properly touch the ground between the vertices.Inorder to avoid this it is better to use only mid or high camera angle for player's eye view.Avoiding low angle helps to hide any gap if happen between the foot of character and ground.

CONCLUSION

The dynamic generation of terrain and characters in a game enhance the reality and dynamicity needed for a player .It also helps to create endless possibilities in 3D virtual reality applications and dynamic simulations.Dynamic generation of terrain elements for different types of games needs to address different issues according to the game type.Dynamic generation of terrain elements and characters is designed to work without consuming too much system resources so that it can work even on home PC and smart phones.

REFERENCES

[1] J Togelius and GN Yannakakis, *A Panorama of Artificial and Computational Intelligence in Games,* University of Copenhagen, Center for Computer Games Research, IT , **2014.**
[2] Chin Hiong Tan, Kay Chen Tan and A Tay, Dynamic Game Difficulty Scaling using Adaptive Behavior-Based AI, *Institute for Infocomm Research, Agency for Science Technology & Research*, Singapore, **2011.**
[3] *www.keralatourism.org*
[4] Image courtesy :The Animator's Survival Kit' by Richard Williams
[5] *www.gamesloth.com*