ALGORITHMS FOR RUN TIME TERRAIN DEFORMATION

STATUS REPORT

SBIR Phase II Contract FA8650-05-C-6537 Topic AF04-064 Sep. 8, 2005

Prepared for AFRL/HEAE, Air Force Research Laboratory
Attn: Steve Stephens, 6030 South Kent Street, Mesa, AZ 85212-6061
Tel: (480) 988-6561 ext. 146 e-mail: steve.stephens@mesa.afmc.af.mil

Prepared by Computer Graphics Systems Development Corp.
Attn: Roy Latham, 2483 Old Middlefield Way #140, Mtn. View CA 94043
Tel: 650-903-4922 Fax: 650-967-5252 e-mail: rwl@cgsd.com SkypeID: rlatham001

Project Abstract

The goals of Phase II are to develop and implement algorithms for a real time mission rehearsal simulation which will deform the terrain database to match target data. A correction function c(x,y) that is added to every vertex in view. The correction function must (1) adjust the terrain surface to meet the specified features, (2) appear smooth and continuous so that the adjustments appear natural, and (3) do not distort aspects of the database that must be preserved. The implementation is to be in C++ and compatible with Open Scene Graph. The code will be placed in the public domain in keeping with an open source philosophy.

• *Previously Completed Work*

▪The Phase II contract was signed on April 8, 2005 and work started immediately. ▪ A kickoff meeting was held in May '05, and the program plans were reviewed. ▪ Don Burns, one of the originators of Open Scene Graph was added to the project team.

• *Work Accomplished This Reporting Period*

This past month was quite productive. Don Burns completed 80% of the software module for reading shapefiles into Open Scene Graph (OSG). SHAPE is a standard format developed by mapping community for representing graphs objects, including point, line, and polygonal data. We have selected SHAPE as the format for importing target data into the dynamic terrain system. The main database would be modified in real time to match the target data in the shapefile.

CGSD had written a shapefile reader for a previous project, however it was written using the Microsoft standard object classes in C++. Don rewrote the code to eliminate the proprietary dependence, and also to interface it to OSG.

The completed shapefile input module for OSG will be posted to the OSG website for public use as soon as testing is complete.

Separately, P.Y. has updated the program description language (PDL) drafts for the terrain algorithms and the for the standalone program that will be used to test implementations of the algorithms. These were originally drafted in Phase I and are now needed as the basis for the next phase of implementation.

The revised versions are presented below.

Complete the first draft of PDL
—Software design in Program Design language

This month we completed a draft of PDL for the software design that (1) has linked Open Scene Graph (OSG) with the terrain correction, (2) has taken into consideration of layers of database, (3) has taken into consideration of classified tiles on a layer. The PDL code is included in this report.

Assumptions

1. A list of targets is given at the time when a datablock is retrieved.
2. A datablock is composed of one or more databases.
3. A database is layered.
4. Each layer of a database is tiled, and some tiles have classified information.
5. Classified tiles are considered more accurate than the corresponding non-classified tiles.
6. Layers of a database include terrain layer, cultural layer, special feature layer, and targetable feature layer.

OSG Interface

Terrain Correction or Deformation module is to be an Open Scene Graph (osg) plug-in.

Get_Datablock
```
	If a file is an Openflight file
		OSGread_openflight_file
		OSGparse_openflight_file
		Main
	Else if a file is a Shapefile
		OSGread_shapefile
		OSGparse_shapefile
		Main
	Endif
```
End Get_Datablock

Terrain Correction or Deformation

Main-
For each datablock of concern
For each database of the datablock
    If a layer is a terrain layer, then
        For each target T on the list of targets for the datablock of concern
            Calculate the height differences of vertices of the target (Cal_Height_Diff(T))
            Find the maximal height difference, say H (Cal_HeightDiff_Max(T))
            If H is not zero
                Place the target in a new list of target-need-correction(Add_To_TNClist)
            Endif
        Endfor

        Group targets on the new list of target-need-correction (Target_Group(list))
        For each group G of targets-need-correction
            Determine the Gaussian correction function (Cal_A(G)) (Cal_B(G)) (Cal_R(G))
            Correct the heights of terrain points around the target accordingly (Correct_Height)
            Record the correction into the terrain patch
        Endfor
        Treat level of details issue (LOD_treatment)
        Retrieve neighboring datablocks if necessary
    Endif
Endfor
Endfor
End Main

Cal_Height_Diff(T) –Calculates the difference of heights between a vertex of the target T and its corresponding terrain point
    Do for each vertex of the target T
        Say the vertex' coordinate is Xt, Yt, Zt
        Identify the tile the vertex is in, say tile TILE(I,J)
        If TILE(I,J) has a corresponding classified tile, then
            Use the classified tile CTILE to
            Find the corresponding terrain point Xt, Yt, Z (interpolate if exact Xt, Yt not found)
        Else
            Use tile TILE to
            Find the corresponding terrain point Xt, Yt, Z (interpolate if exact Xt, Yt not found)
        Endif
        Diff= Zt – Z for the vertex
    Enddo
End Cal_Height_Diff

Cal_HeightDiff_Max(T) – Calculates the max of the height differences for a target

```
    Set Maxdiff = 0
    Do for each vertex of a target
        Take absolute value of the height difference for the vertex, abs(hightdiff)
        Max diff = max(Maxdiff, abs(heightdiff))
    Enddo
```

End Cal_HeightDiff_Max

Add_To_TNClist – Adds a target to the list of Target Need Correction

```
    Assign the pointer for the TNClist to the target
```

End Add_To_TNClist

Target_Group(list)– Groups targets of a given list using some predetermined parameter(s), for example, distance D

```
For a group G
    Select a target as the base target, and call the group for the base target G1
    Do for each target
        Calculate distance between the base target and the target
        If the distance is greater than D
            Place the target into group G2
        Else
            Place the target into the same group as G1
        Endif
    Enddo
    Use G2 as a new group G, repeat the process
Endfor
```

End Target_Group

Cal_A(G) – Calculates a for the Gaussian function for a group G of targets

```
    A = max of the pairwise distance among targets in the group G, for example
```

End Cal_A

Cal_B(G) – Calculates b for the Gaussian function for a group G of targets

```
    B = max of height differences of targets in the group G, for example
```

End Cal_B

Cal_R(G) – Calculates r for the Gaussian function for a group G of targets

```
    R = diameter of the circular hull for the group G of targets
```

End Cal_R

Correct_Height(G) – Correct the terrain that found mismatch with the targets in group G

```
    Do for each terrain point fall in the area defined by the Gaussian function
        Add the Gaussian correction value to the height
    Enddo
```

End Correct_Height

LOD_treatment – Provides Level of Details treatment

Generate level 1 terrain patch by combining 2 pixels
Generate level 2 terrain patch by combining 4 pixels
Generate level 3 terrain patch by combining 8 pixels
Generate level 4 terrain patch by combining 16 pixels

End LOD_treatment

Synthetic Test Database Generation

Build_Terrain_Array (N,A)

Builds an N x N array of synthetic terrain elevation data. N is a power of 2. Each point in the array is the elevation of the terrain in feet at that point. The data is random with the special frequencies having a spectrum that falls proportional to the frequency. The lowest frequency has a magnitude of 1000 feet.

Check if N is a power of two.
Set all the values in the array to zero.
Compute Number_of_Iterations = log2 (N). so a 32 x 32 array will take 5 passes
For MPASS = 0 to Number_of_Iterations – 1
Initialize A = 1000
Pick four random numbers R1, R2, R3, R4
Fill_Terrain_Square (0, A, 0, 0, N, R1, R2, R3, R4)

END Build_Terrain_Array

Fill_Terrain_Square (MPASS, A[N, N], LLx, Lly, M, R1, R2, R3, R4)

This adds higher frequency data to an M x M subgrid within an existing N x N array A of terrain gridposts. R1 .. R4 are four random numbers. The x,y coordinates of the lower left corner of the subgrid is LLx, Lly. MPASS controls the frequency of the noise added to the grid, with finer subdivision on each pass.

Check if N is a power of two and MPASS > 1

If M is < 2 then RETURN

URX = LRX + M
URY = LRY + M

Scale the four random numbers R1, R2, R3, R4 so each in the range - 500/(2**MPASS) to +500/(2**MPASS)

Compute the plane parameters for the triangle in the upper right

```
R1----R2
|     /
R3
```

DXU = (R2 – R1)/M
DYU = (R3 - R1)/M

Compute the parameters for the triangle in the lower right

```
        R2
     /   |
R3----R4
```

DXL = (R4 – R3)/M
DYL = (R2 – R4)/M

```
FOR IY = 0 TO M-1
FOR IX = 0 TO M-1

IF IX < IY THEN A(LRX+IX,LRY+IY) += R3 + DXU*IX + DYU*IY
         ELSE A(LRX+IX,LRY+IY) += R3 + DXL*IX + DYL*IY
END for IX
END for IY
```

Divide the Array into four subgrids. Pick 9 random numbers S1 .. S9. They define the values at the corners of the four subgrids as follows:

```
S1 +++ S2 +++ S3
S4 +++ S5 +++ S6
S7 +++ S8 +++ S9
```

Fill each of the subgrid using MPASS<=MPASS + 1
N2 = (URX – LLX)/2
Fill_Terrain_Square (MPASS, A[N, N], LLx, Lly, N2, S7, S5, S4, S8)
Fill_Terrain_Square (MPASS, A[N, N], LLx, Lly+N2, N2, S4, S2, S1, S5)
Fill_Terrain_Square (MPASS, A[N, N], LLx+N2, Lly+N2, N2, S5, S3, S2, S6)
Fill_Terrain_Square (MPASS, A[N, N], LLx+N2, Lly, N2, S8, S6, S5, S9)
End Fill_Terrain_Square

Correct_Terrain (N, A, F, T)

A is the N x N array of terrain gridposts
F is the correction function
T is the vector containing NT target points

This routine adds the correction function to the terrain grid posts.

FOR IX = 0, N
FOR IY = 0, N

A(IX, IY) += F(IX,IY, T)

END for IY
END for IX

End Correct_Terrain

Display_Terrain (N, A)
A is the N x N array of terrain gridposts. Scale 300 feet
V1, V2, V3, V4 are 3D vertices, each with (x, y, x) coordinates.

Set the viewpoint so as to see the terrain in perspective. Put the eye point at (-100000, -100000, 2000). Looking along the diagonal.

Set the window matrix to 40 x 50 degrees

Initialize the OpenGL pipeline for green terrain triangles with 20% ambient and 80% direct sun from 60 degrees above the north horizon

Display two 3D triangles for each gridpoint, the two triangles to the upper right.

For IX = 0, N-1
For IY = 0, N-1

Scale up IX by SCALE
V1 = IX, IY, A(IX, IY)
V2 = IX, IY+1, A(IX, IY+1)
V3 = IX+1, IY+1, A(IX+1, IY+1)
V4 = IX+1, IY, A(IX+1, IY)
Display_Triangle (V1, V2, V3)
Display_Triangle (V1, V3, V4)

End for IY
End for IX

End Display_Terrain

Display_Point (V)
Define as constants the six points that define an up-tetrahedron conjoined with a down-tetrahedron. The top vertex is V1, the bottom vertex is V2, and V2 through V5 surround the point.

V1 = (0,0,1)
V2 = (-1,-1,0)
V3 = (-1,1,0)

V4 = (1,1,0)
V5 = (1,-1,0)
V6 = (0,0,-1)

Display a point, such as a target point, on the screen

Set the color to red.

Scale the x, y coordinates with SCALE

Display_Triangle (V+V1, V+v2, v+v3)
Display_Triangle (V+V1, V+v3, v+v4)
Display_Triangle (V+V1, V+v4, v+v5)
Display_Triangle (V+V1, V+v5, v+v1)
Display_Triangle (V+V6, V+v3, v+v2)
Display_Triangle (V+V6, V+v2, v+v5)
Display_Triangle (V+V6, V+v5, v+v4)
Display_Triangle (V+V6, V+v4, v+v3)

End Display_Point

Display_Triangle (V1, V2, V3)

V1, V2, V3 are 3D vertices, each with (x, y, x) coordinates.
Display_Line (V1, V2)
Display_Line (V2, V3)
Display_Line (V3, V1)

END Display_Triangle

Main_TestDatabase

Get the parameter, N for the grid size, the number of targets
Get an integer IRAND to seed the random number generator
Initialize the random number generator
Clear the screen
Draw a 2-D box and fill it with a blue background
Print the name of the company, the program, the date, the time, and N as a caption
Build_Terrain_Array (N,A)
Generate the targets points T
Display_Terrain (N, A)
For each T(i) Display_point(Ti)

Ask (What correction function?) get (Function Number) and a function parameter
Write the function selected in the caption
Correct_Terrain (Function Number, T)
Display_Terrain
For each T(i) Display_Point(Ti)

Wait for space bar to continue with a new parameter or Q to quit

END Main_TestDatabase

• *Summary of Status*

The project is on schedule. The status of tasks is summarized below:

ID	Description	9/8/05
Task 1	Research & verify the timeliness of the full-scale Algorithm/technique	90%
Task 2	Verify the accuracy of the full-scale algorithm or technique	60%
Task 3	Design, code and test the full-scale algorithm	30%
Task 4	Develop a web site for the release of open source code	30%
Task 5	Examine the compatibility of the open source code with the existing IG hardware	0%
Task 6	Demonstrate the prototype	0%
Task 7	Write Interim Report(s)	20%
Task 8	Write Final Report and Summary Report 0%	0%

• *Problems*

No significant problems or information that might impact schedule have been encountered in this reporting period.

• *Interim Results*

There are no interim results to report in this period.

• *Recommendations and Proposals*

There are no recommendations or proposals as a result of efforts in this reporting period.

• *Summary of Future Plans*

We expect to complete the shapefile reader in the coming month, and to post it on the web for public use. Implementation of the standalone test program will then be started.