Relief mapping for urban and natural environments rendering

D. Janković* and Ž. Mihajlović **
* AVL/AST -- Advanced Simulation Technologies, Zagreb, Croatia
** University of Zagreb, Faculty of Electrical Engineering and Computing/
Department of Electronics Microelectronics, Computer Science and Intelligent Systems,Zagreb, Croatia
danijel.jankovic@avl.com, zeljka.mihajlovic@fer.hr

Abstract - Terrain models typically contain huge amount of data so they are very time consuming for visualization purposes. This especially comes to the forefront when urban environments are included. The main compromise in representation of the complex environments is between achieved quality and time consumption. With the simple texture representation of complex environments we will accomplish fast application, and with the large polygonal meshes, high quality of the rendered scene.

In this paper we propose rendering of urban and natural environments using parallax and relief mapping. This approach combines benefits of the rendering of polygonal meshes and texture approach. Thereby, in the proposed approach improved quality on the one side and increased speed on the other side is combined. The applicability of the method is demonstrated trough parallax and relief mapping within the Irrilicht open source graphics engine. The shaders programs were made with the GLSL shader language. As the result, the tests were made to determine the possible usage of parallax and relief mapping in the display of natural and urban environments.

Keywords: terrain rendering, urban environments, relief mapping, parallax mapping

I. INTRODUCTION

There are several ways of terrain visualization, the first and oldest is the so called top down view, used for centuries on cartographic maps, while the second being perspective rendering used in modern computer graphics. Terrain has always been an important factor in military simulations being used to accustom pilots to hostile territory as well as to help commanders make important decisions.

For a typical terrain rendering application, in some simulation scenario, it is important to accomplish an acceptable frame rate, and at the same time insure high quality of scene representation. If the terrain is observed from closer distances, only a small part of it should be considered but with a high level of detail. For the far point of view details are not visible anyway, therefore texture representation could be used instead of some distant complex object. The main problem is between those two extreme cases, when larger part of terrain is observed but details are not so distant to be neglected. For flight simulation purposes representation of large or even huge terrains, various techniques are used to reduce graphic overload [7], [8]. Usually these techniques are based on managing LOD of polygonal mashes depending on view position.

There are several main components that could cause decrease of application performances: bandwidth, central processing unit CPU, and graphics processing unit GPU. The CPU loads and identifies the terrain data, does the necessary transformations for creating a mesh of points which is then sent to the GPU for additional processing and rendering. In modern 3D graphics applications it is necessary to display large, photorealistic terrains with a large amount of detail.

The problem is always the computing power available to process all the required data. More specifically, the terrain in flight simulators extends for hundreds if not thousands of kilometers, with satellite images added to it as textures for enhanced realism. This is fine if the user stays at a relatively large distance from the terrain, when getting closer the fact that the terrain is flat becomes more obvious. The first and most common solution to this is adding textured 3D polygon mesh as buildings on top of the texture. The problem with this approach is the additional geometry for the system to process, especially for dense urban areas.

The main idea in this paper is to reduce amount of geometric data but to maintain perception of objects' height. Therefore, in this paper we propose the techniques of parallax and relief mapping in an attempt to determine their potential in decreasing the resource cost for terrain rendering.

II. PARALLAX MAPPING

Parallax mapping is also known as offset mapping or virtual displacement mapping and is an enhancement of the well known bump and normal mapping techniques. The term parallax refers to the difference in the apparent position of an object viewed along two different lines of sight, and is measured by the angle or semi-angle of inclination between those two lines as presented in Zinc [2].

This work has been carried out within the project 036-0362980-1921 Computing Environments for Ubiquitous Distributed Systems funded by the Ministry of Science, Education and Sport of the Republic of Croatia.

Figure 1. Parallax mapping on the natural terrain, a bumpy effect on the surface is observable.

The final result of the parallax mapping technique is enhanced depth and irregularity, there are deformations of the terrain similar to bumps giving the illusion of a jagged terrain (Figure 1), and thus greater realism of the observed texture is achieved.

The parallax mapping technique works by offsetting the texture coordinates at a point on the rendered polygon by a function of the view angle in tangent space (the angle relative to the surface normal) and the value of the height map at that point. At steeper view-angles, the texture coordinates are displaced more, giving the illusion of depth due to parallax effects as the view changes. The texture offset calculation is relatively simple:

$$T_n = T_0 + h_{sb} \cdot \frac{V_{\{x,y\}}}{V_{\{z\}}}. \qquad (1)$$

Where T_n represents the new texture coordinates, T_0 the original texture coordinates, h_{sb} is the scaled and biased height, and $V_{\{x,\,y,\,z\}}$ is the normalized eye vector. An improvement of this technique is *Parallax mapping with offset limiting* which limits the maximal texture offset resulting in fewer errors caused by texture overlapping, [5]. The modified equation is:

$$T_n = T_0 + h_{sb} \cdot V_{\{x,y\}}. \qquad (2)$$

It does have flaws as it cannot account for occlusion and self-shadowing. For that reason the relief mapping technique is further investigated in this paper.

GlSL vertex shader code for parallax mapping is following:

```
uniform vec3 LightPos, EyePos;
varying vec3 LightDir, EyeDir;
varying vec2 TexCoord;
attribute vec3 rm_Tang, rm_Binor;
void main()  {
    vec3 pos =  normalize(vec3(gl_ModelViewMatrix * gl_Vertex));
    vec3 Light = LightPos - pos;
    vec3 Eye   = EyePos - pos;
    vec3 Tangent = normalize(vec3(gl_NormalMatrix * rm_Tang));
    vec3 Normal = normalize(vec3(gl_NormalMatrix * gl_Normal));
    vec3 Binormal =normalize(vec3(gl_NormalMatrix * rm_Binor));
    LightDir.x = dot(Light, Tangent);
    LightDir.y = dot(Light, Binormal);
    LightDir.z = dot(Light, Normal);
    EyeDir.x = dot(Eye, Tangent);
    EyeDir.y = dot(Eye, Binormal);
    EyeDir.z = dot(Eye, Normal);
    TexCoord = gl_MultiTexCoord0.xy;
    gl_Position = ftransform();}
```

For each vertex light vector Light and eye vector Eye are transformed in TBN space. Fragment shader for each pixel calculate appropriate texture element according to the equation (2) where variable tHeight denotes h_{sb}:

```
uniform vec4 ambientC, diffuseC, specularC;
uniform sampler2D ColorTex, NormTex;
uniform float s, b, a;
varying vec3 LightDir, EyeDir;
varying vec2 TexCoord;
void main(void) {
  float d = length(LightDir);
  vec3 tLightDir = normalize(LightDir);
  vec3 tEyeDir = normalize(EyeDir);
  float attenu = 1.0/(a * d);
  float tHeight = texture2D(NormTex, TexCoord).w * s + b;
  vec2 tTexCoord = TexCoord + (tHeight * tEyeDir.xy);
  vec3 tNormal = normalize(vec3(texture2D(NormTex,
tTexCoord)) * 2.0 -1.0);
  float fNDotL = dot(tNormal, tLightDir);
  vec3 fvReflection=normalize(((2.0*tNormal)*fNDotL)-tLightDir);
  float fRDotV  = max(0.0, dot(fvReflection, tEyeDir));
  vec4 fvBaseColor = texture2D(ColorTex, tTexCoord);
  vec4  fvTotalAmbient = ambientC * fvBaseColor * attenu;
  vec4  fvTotalDiffuse = diffuseC * fNDotL* fvBaseColor* attenu;
  vec4  fvTotalSpecular = specularC * (pow(fRDotV, 25.0))*attenu;
  gl_FragColor= fvTotalAmbient+fvTotalDiffuse+fvTotalSpecular;
}
```

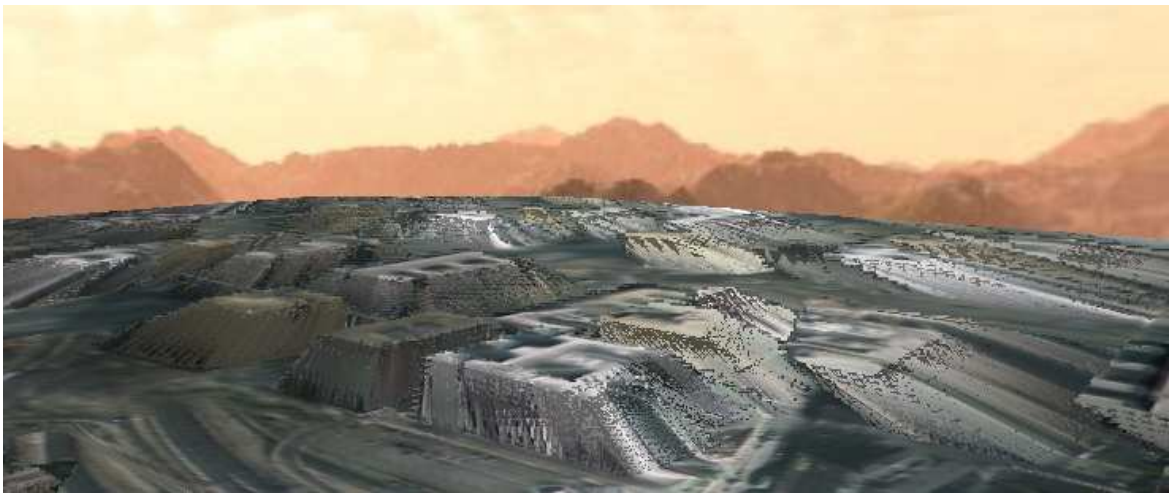

Figure 2. Relief mapping at small angles.

III. RELIEF MAPPING

Relief mapping uses the programmability of modern GPUs to implement a pixel-driven solution to relief texture mapping. For the relief mapping beside ordinary texture an additional 32 bit texture with normal vectors stored in the RGB channels and the height map stored in the α channel of the image should be prepared [6].

This technique produces the parallax effect along with self-occlusion, self-shadowing, and view-motion parallax. It is, in essence, a short distance ray trace computed in the pixel shader. The input to the algorithm is the depth characteristic of the texture and the view vector. The pre-deformation solves the problems of visibility and filling in any potential blanks. The output picture of the algorithm is simply used as an input to the standard texture mapping technique.

The two parameters important to note here are linear search depth and binary search depth. Binary search is used in determining the intersection between the view ray and the depth characteristic while the linear search is used to determine the first point of intersection with the depth characteristic [1].

There are also some other approaches in determining the ray intersection with surface [3]. The flaws of the technique come into being when viewing the texture from small angles, resulting in a fish eye effect on the surface because of intersection miscalculations, presented in Figure 2. The solution to this is using cone step mapping.

In vertex shader for relief mapping tangent, normal and binormal are transformed in eye space, and in the fragment shader short distance ray tracer is programmed. The main part of fragment shader is:

```
float ray_intersect(in vec2 dp, in vec2 ds) {
   //const int linear_search_steps=20;
   //const int binary_search_steps=5;
float depth_step=1.0/float(linear_search_steps);

float size=depth_step;  // size of search window
float depth=0.0;  // current depth position
float best_depth=1.0; // best match for depth
  // intersection of ray with object
  for( int i=0;i<linear_search_steps-1;i++ ) {
      depth+=size;
      vec4 t=texture2D(reliefMap,dp+ds*depth);
      if (best_depth>0.999)   // if no depth found
      if (depth>=t.w)
        best_depth=depth;   // best depth
  }
  return best_depth;
}
```

IV. RESULTS

The techniques mentioned above were implemented inside the Irrilicht open source engine [4], with the ability to modify the offset value for parallax mapping, and the linear search steps value for relief mapping. The shaders themselves are simple implementations of parallax and relief mapping written in the OpenGL shading language (GLSL) as presented in II and III.

The main problem with reproducing urban environments is the lack of city height maps. It would certainly be better to use real height maps if they are available. But, if the real height map is unavailable, the idea is to use specular lighting approximation to simulate height maps of urban environments. For parallax mapping specular lighting was used to simulate the height maps (Figure 3) and results are shown to demonstrate the usefulness of this approximation. Approximation of a height map according to the given texture map of an urban environment is presented in Figure 4.

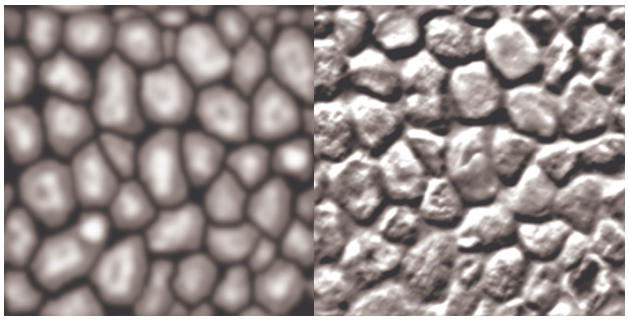

Figure 3. Texture height map and specular lighting approximation.

Figure 4. Texture and heightmap approximation of urban environment.

Using the mentioned approximation of a height map the following results were produced with the usage of parallax mapping. For evaluation purposes the testing configuration was: CPU: AMD Athlon X2 5000 Mhz, RAM: 2GB DDR2 800 MHz, GPU: ATI 4670 512 MB DDR3 OS: Windows XP SP2.

Firstly, we compare the rendering speed of urban and natural environments rendering using ordinary texture mapping and parallax mapping. If parallax mapping is applied the mountain range appears higher than the rest of the terrain and deeper areas are shown correctly. The performance cost was only 14% since parallax mapping performs very fast on modern GPU-s.

Relief mapping, when compared to standard texture mapping is more time demanding. As seen from Table 1. relief mapping reduces performances but significantly improves 3D perception (Figure 5). When relief mapping is applied, the terrain is realistically bumpy with correct shadows and surface irregularities. We should also be aware that static pictures do not reveal fine details as interactive movement around the shown terrain.

TABLE 1. COMPARISON BETWEEN THE TEXTURE MAPPING AND RELIEF MAPPING.

Test	Texture mapping	Relief mapping
Urban environment texture resolution 512 x 512 repeated over the terrain	176 fps	44 fps
Urban environment, texture resolution 1024 x 1024	196 fps	56 fps
Natural environment, texture resolution 1280 x 1280	170 fps	60 fps

Finally, we present performance comparison between texture mapping, polygonal mesh rendering with different LODs, parallax mapping and relief mapping (Table 2). Parallax mapping is a common technique in modern video games and its usage is well documented making it easy to implement in any API. It can be used for high altitude simulations, applied to the terrain previously generated by polygons for enhanced realism and present a desirable effect in any 3D environment (Figure 1).

TABLE 2. PERFORMANCE COMPARISON BETWEEN VARIOUS RENDERING CONCEPTS

Test 1024 x 1024 natural texture	Performance
Texture mapping	824 fps
Polygon rendering (1 LOD, 3195724 polygons)	5 fps
Polygon rendering (3 LOD, 199724 polygons)	62 fps
Polygon rendering (5 LOD, 49940 polygons)	170 fps
Parallax mapping	716 fps
Relief mapping	55 fps

The performance cost for parallax mapping is low, averaging in a 14.06% decrease compared to texture mapping. Taking into account the high frame rates produced by modern computer systems this is a barely noticeable slowdown. However parallax mapping cannot replace 3D polygons even at medium distance because of its shortcomings, lack of self occlusion, noticeable artifacts at smaller distances, and from small viewing angles due to the approximation of the texture offset.

Relief mapping is a technique which has great potential, creating a convincing 3D surface from a previously flat image. It elevates the higher parts of the terrain and produces correct shadowing (Figure 5). Its drawbacks however are a high processing cost and errors which occur when the surface is viewed from a small angle (Figure 2).

Relief mapping can be used on natural and urban terrain at medium view distances with great effect, producing a realistic elevation effect (Figure 5), especially if height maps of the desired terrain are available. Usage of relief mapping from a large distance is not recommended because of the large processing cost. In case of natural terrain without many buildings it is recommended to use the standard polygon rendering with LOD algorithms because relief mapping cannot be used for extreme close-ups such as flybys through canyons.

However it is important to note that relief mapping has demonstrated to be more efficient than standard polygon rendering without LOD algorithms when used on natural terrains. The usage of relief mapping results in a 69.75% performance drop, making it unusable in many modern complex applications and on older systems. The technique is still being optimized and it is likely to be used in modern graphic applications in the near future.

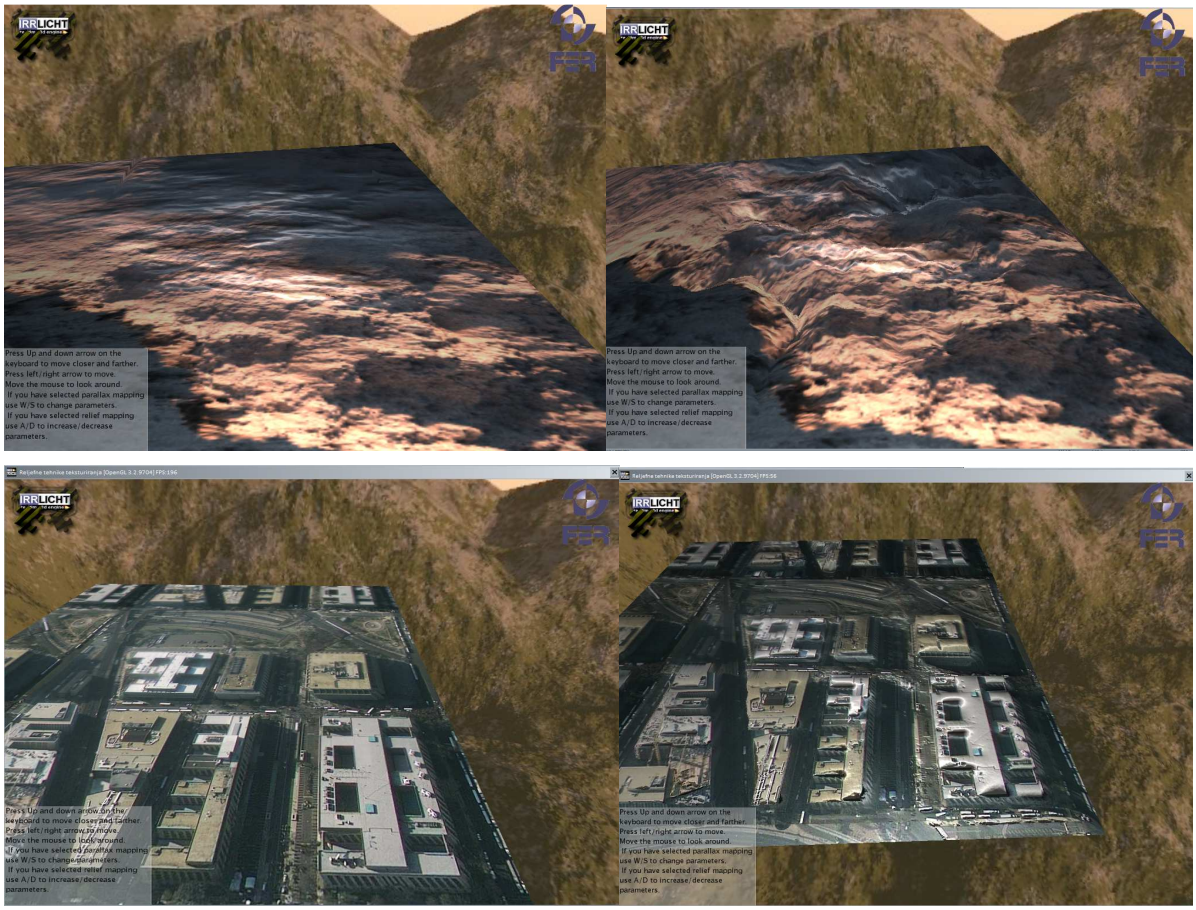

Figure 5. Texture mapping (left) and relief mapping (right).

V. CONCLUSION

As is observable from the results, the approximation of the terrain using parallax mapping is an efficient and cost effective technique to simulate the depth of a surface which can be easily implemented using pixel and vertex shaders. It produces bumps and indentations in the terrain elevating the realism factor by a significant margin. Specular images used for height map approximation can be created very easily, and the parameters of the image can be modified to suit specific visual needs. Relief mapping is appropriate at medium view distances on terrain.

From the overall analysis we can conclude that parallax and relief mapping has potential in natural and urban terrain rendering at the appropriate viewing distances for the representation of urban and natural environments.

REFERENCES

[1] F. Policarpo, M.M. Oliveira, and J.L. Comba "Real-time relief mapping on arbitrary polygonal surfaces", In Proceedings of the 2005 Symposium on interactive 3D Graphics and Games. I3D '05. ACM, 2005, pp. 155-162. doi: 10.1145/1053427.1053453

[2] J. Zinc "A Closer Look at Parallax Occlusion Mapping", http://www.gamedev.net/columns/hardcore/pom/ Accessed January 2010.

[3] E. Risser, S. Musawir, and P. Sumanta P, "Interval Mapping. University of Central Florida" Technical Report. Available online http://graphics.cs.ucf.edu/IntervalMapping/images/IntervalMapping.pdf.

[4] Irrilicht open source engine 2010, http://irrlicht.sourceforge.net, Accessed 2010.

[5] T. Welsh, "Parallax mapping with offset limiting: a per pixel approximation of uneven surfaces". http://pds1.egloos.com/pds/1/200603/10/62/parallax_mapping.pdf, Accessed Jan 2010.

[6] F. Policarpo and M.M. Oliveira, Relief mapping of non-height-field surface details. In Proceedings of the 2006 Symposium on interactive 3D Graphics and Games, March 14 - 17, 2006. I3D '06. 2006, pp. 55-62. DOI= http://doi.acm.org/10.1145/1111411.1111422

[7] J. Schneider and R. Westermann, "GPU-friendly high-quality terrain rendering". Journal of WSCG, 14 (1-3): 2006, pp. 49–56.

[8] F. Losasso, H. Hoppe, Geometry clipmaps: Terrain rendering using nested regular grids ACM Trans. on Graphics (Proc. SIGGRAPH 2004), Vol. 23, No.3, 2004, pp. 769-776.