한국컴퓨터그래픽스학회 Vol. 12, No. 4, P. 1~5

하이트필드 맵을 이용한 회화적 질감 표현

류승택
한신대학교 컴퓨터정보소프트웨어학부
stryoo@hs.ac.kr

Surface Detailed Painterly Rendering Using Heightfield Map

SeungTaek Ryoo
Division of Computer · Information and Software, HanShin University

요 약

본 논문에서는 하이트필드 맵을 이용한 회화적 질감 표현 방법을 제시한다. 회화적 렌더링된 결과물의 하이트필드 맵을 이용하면 회화작품의 질감을 표현할 수 있다. 이를 위해 하이트필드 맵을 바탕으로 노말 매핑과 디스플레이스먼트 매핑 방법을 이용하여 회화적 렌더링을 구현하였다. 제시된 방법은 GPU 프로그래밍을 이용하여 물체의 세부적인 표면을 실시간으로 회화적 렌더링할 수 있어 3차원 가시화나 게임엔진에 응용할 수 있다.

Abstract

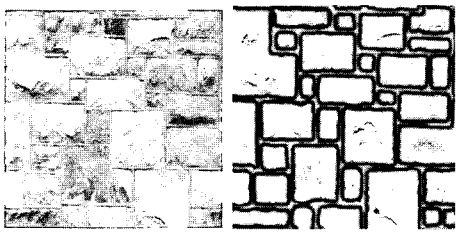
This paper introduces the surface detailed painterly rendering using heightfield map. To do this, we implement painterly rendering using normal mapping and displacement mapping method by heightfield map. The suggested method can apply to the 3D visualization program and game engine for representing the surface detailed realtime rendering using GPU programming

키워드 : 회화적 렌더링, 디스플레이스먼트 매핑, 하이트필드
Keywords : painterly rendering, displacement mapping, heightfield

1. 서론

최근 그래픽 하드웨어를 이용한 GPU 프로그래밍이 가능하게 되면서 실시간 렌더링에 대한 연구가 활발히 이루어지고 있다[1-6]. 예전까지 불가능하게 여겨졌던 광선추적법, 레디어서티와 같은 고급 렌더링 방법들이 GPU를 가진 그래픽 하드웨어의 급속한 발전으로 실시간 렌더링이 가능하게 되었다. 본 논문에서는 GPU 프로그래밍을 이용한 회화적 질감 표현을 제시한다. 이를 위해 하이트필드 맵(Heightfield Map)을 이용하여 노말 매핑(Normal Mapping)과 디스플레이스먼트 매핑(Displacement Mapping) 방법을 구현하였다. 노말 매핑 방법은 하이트필드 맵을 노말 맵으로 변환하여 물체의 노말을 노말맵의 노말로 대치하여 표현함으로써 적은 비용을 가지고 물체의 재질을 표현할 수 있다는 장점이 있다. 그러나 실제 물체의 위치값은 그대로 남아 있어 물체의 실루엣이나 물체의 세부적 표현을 할 수 없다는 단점을 가지고 있다. 이를 보완하기 위해 실제 물체의 위치값을 하이트필드 맵을 이용하여 변환하여 물체의 디테일을 표현하는 디스플레이스먼트 매핑방법이 소개되었다[7]. 본 논문에서는 회화적 렌더링을 위한 노말 매핑과 디스플레이스먼트 매핑 방법을 OpenGL과 Cg를 이용하여 GPU 프로그래밍으로 구현하였다. 제시된 방법은 회화적 렌더링의 세부적인 질감을 표현할 수 있으며 GPU 프로그래밍을 이용하여 물체의 세부적인 표면을 실시간으로 렌더링할 수 있어 3차원 가시화나 게임엔진에 응용할 수 있다.

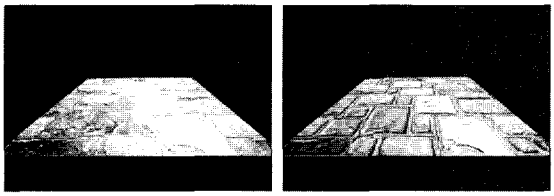
2. 하이트필드 맵을 이용한 노말 매핑



<그림 1> 데칼 맵과 하이트필드맵

하이트필드 맵을 이용하여 물체 표면의 질감을 표현하기 위해서는 노말맵으로의 변환 과정(노말매핑)이나 실제 물체 표면의 높이 값을 변환하는 방법(디스플레이스먼트 매핑)이 필요하다. 본 연구에서는 노말 매핑 방법과 디스플레이스먼트 매핑 방법을 이용하여 물체의 세부묘사를 하였다.

그림 1은 실제 물체에 매핑될 데칼 맵과 물체의 높이 값을 표현하는 하이트필드 맵을 보여주고 있다. 하이트필드 맵을 이용하여 노말 매핑을 하기 위해서는 하이트필드 맵을 노말맵으로 변환하여야 한다. 이렇게 생성된 각 텍셀의 노말을 물체의 밝기값을 구할 때 물체 표면의 노말 벡터로 대치하여 사용하면 세밀한 표면의 효과를 줄 수 있다. 그림 2는 노말 매핑 방법을 이용한 벽돌의 세부 표면을 렌더링한 결과이다.

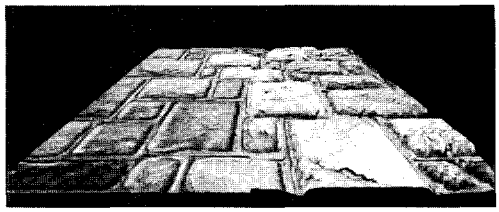


(a) 텍스춰 매핑 (b) 노말 매핑(범프 매핑)

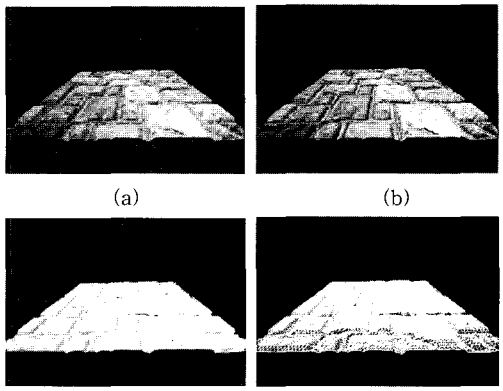
<그림 2> 노말 매핑에 의한 세부적 표면 렌더링

3. 하이트필드 맵을 이용한 디스플레이스먼트 매핑

노말 매핑 방법은 적은 비용으로 물체의 표면을 렌더링하여 보여줄 수 있으나 실제 물체의 위치값은 변화하지 않아 물체의 실루엣이나 줌인된 물체의 세부적 표면을 보여줄 수 없다는 문제점을 가지고 있다. 이를 해결하기 위해서는 하이트 필드 맵을 이용하여 실제 표면의 위치로 변환하는 디스플레이스먼트 매핑 방법이 필요하다. 그림 3은 하이트필드맵를 이용하여 구현된 노말 매핑(왼쪽)과 디스플레이스먼트 매핑(오른쪽) 방법을 비교하여 보여주고 있다.



<그림 3> 노말 매핑(왼쪽부분)과 디스플레이스먼트 매핑(오른쪽부분) 비교

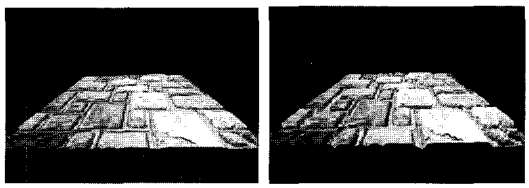


(a) (b)

(c) b의 데칼맵 제거후 (d) b의 선구조 형상

<그림 4> 디스플레이스먼트 매핑에 의한 세부적 표면 렌더링
(a: 디스플레이스먼트 매핑, b: 디스플레이스먼트 + 노말 매핑)

그림 4는 디스플레이스먼트 매핑을 통해 렌더링된 결과영상을 보여주고 있다. 그림 4-a는 하이트필드맵을 이용하여 실제 표면의 위치로 변환된 물체의 버텍스 노말을 사용하여 렌더링된 영상을 보여주고 있으며, 그림 4-b는 디스플레이스먼트 매핑과 노말 매핑을 이용하여 렌더링된 영상을 보여주고 있다. 그림 4에서 볼수 있듯이 노말 매핑 방법과 디스플레이스먼트 매핑 방법을 함께 사용한 결과 영상이 물체의 세부적 표면을 가장 잘 나타냄을 알 수 있다.



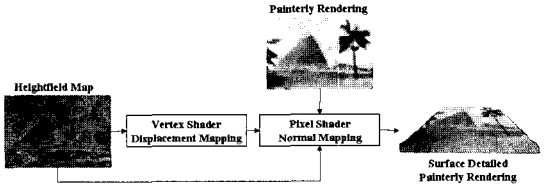
(a) 높이 조절 (왼쪽: 0.3, 오른쪽: 0.7)

<그림 5> 디스플레이스먼트 매핑의 높이

그림 5와 같이 높이값 조절을 통해 물체의 세부적 질

감을 조절할 수 있다. 실제 물체의 버텍스 노말을 사용하면 물체의 쉐이딩 질감이 떨어진다. 이를 보정하기 위해 하이트필드 맵으로부터 픽셀 노말을 구하여 사용한다 (그림 8).

4. 회화적 질감 표현



<그림 6> 회화적 질감 표현 단계

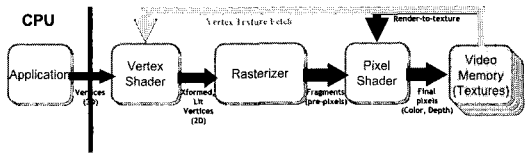
본 논문에서는 노말 매핑 방법과 디스플레이스먼트 매핑 방법을 사용한 회화적 질감 표현 방법을 제안한다. 회화적 질감 표현을 하기 위해서는 회화 작품의 세밀한 질감을 표현하는 하이트 맵이 필요하다. 본 논문에서는 사용자가 에디팅한 가상의 하이트 맵을 사용하였다. 회화적 질감을 표현하는 획득방법에 대한 연구가 향후 연구로 필요하다.

회화적 질감 표현을 하는 단계는 그림 6과 같이 회화적 렌더링된 결과와 회화적 질감을 표현하는 하이트 필드 맵을 입력으로 받는다. 먼저 버텍스 쉐이더에서 하이트필드맵을 이용하여 실제 물체의 높이값으로 디스플레이스먼트 매핑을 한다. 그다음으로 픽셀쉐이더에서 버텍스쉐이더에서 사용한 하이트필드맵을 이용하여 픽셀의 노말을 구하여 노말 매핑한다. 이렇게 렌더링 된 결과는 회화작품의 세밀한 표면 및 질감을 표현 할 수 있다.

그림 9는 노말맵을 이용한 허츠만의 회화적 질감 표현 방법[8]과 본 논문에서 제안된 디스플레이스먼트 매핑방법을 이용하여 회화적 질감을 표현한 결과 영상을 비교하여 보여주고 있다. 제시된 방법이 회화적 작품의 실루엣을 표현할 수 있어 보다 표면을 세부적으로 렌더링함을 알 수 있다.

5. GPU를 이용한 질감 표현

GPU 프로그램을 이용하여 노말 매핑을 적용하기 위해서는 픽셀 쉐이더 부분에 하이트필드 맵을 이용하여 변환한 노말 맵으로 각 픽셀의 노말 값을 변경하여 렌더링하면 된다. 그러나 디스플레이스먼트 매핑을 구현하기위해서는 버텍스 쉐이더에서 텍스춰맵을 읽을 수 있는 특수한 기능을 필요로 한다. 최근에 나온 쉐이더 모델 3.0에서는 이러한 기능(Vertex Texture Fetch)을 제공한다 (그림 7).



<그림 7> GPU의 버텍스쉐이더와 픽셀쉐이더

디스플레이스먼트 매핑은 먼저 버텍스 쉐이더에서 하이트필드 맵을 읽어들여 물체의 위치를 변환하고 픽셀쉐이더에서는 노말맵의 노말을 사용하여 렌더링한다. 버텍스 쉐이더와 픽셀쉐이더에 대한 알고리즘은 표 1과 표2에서 설명하고 있고 본 논문에서 사용한 버텍스 쉐이더와 픽셀 쉐이더는 표3과 표4에서 보여주고 있다.

본 논문에서는 GPU 프로그래밍을 하기 위해 Nvidia Cg와 OpenGL를 이용하였고 그래픽카드로는 Nvidia Geforce 7800과 팬티엄IV 1.81GHz를 사용하였다. 물체의 폴리건의 수가 2만개 이상일때 초당 15플레임을 렌더링할 수 있다.

<표 1> 버텍스 쉐이더

```
Vertex_Shader
Input:  포지션 (x, y, z),  노말, 텍셀 (u, v),
                하이트필드 맵
Output: 변경된 노말, 물체의 텍셀
                시각변환된 포지션, 조명, 하프 벡터
{
   하이트필드 맵을 이용한 포지션의 위치 변환
   변경된 물체의 버텍스 노말 계산
   물체의 포인트의 시각 변환
   조명, 하프 벡터의 시각 변환
}
```

<표 2> 픽셀 쉐이더

```
Pixel_Shader
Input: 시각변환된 포지션, 노말, 조명, 하프 벡터, 텍셀,
              데칼맵, 하이트필드맵
Output:  디스플레이스먼트 매핑이 적용된 물체의 칼라값
{
  하이트필드 맵을 이용한 픽셀 노말 계산
  조명 벡터를 이용한 난반사 계산
  하프 벡터를 이용한 정반사 계산
  계산된 난반사와 정반사를 이용한 물체의 밝기값 계산
  데칼맵의 텍셀값과 밝기값에 의한 최종 칼라값 계산
}
```

6. 결론 및 향후 연구과제

본 논문에서는 물체의 세부적인 표현을 실시간으로 렌

더링하기 위해 GPU 프로그래밍을 이용한 노말 매핑과 디스플레이스먼트 매핑 방법을 제시하였다. 물체의 세부적인 표면을 표현하는 하이트필드 맵을 사용하였다. 제시된 방법은 GPU를 이용하여 물체의 세부적인 표면을 실시간으로 렌더링할 수 있어 3차원 가시화나 게임엔진에 응용할 수 있다.

감사의 글

본 논문은 정통부 및 정보통신연구진흥원의 정보통신선도기반기술개발사업의 연구결과로 수행되었습니다.

참고문헌

[1] Tomas Moller, Eric Haines, Real-Time Rendering, A K Peters, 1999

[2] David Luebke, General-Purpose Computation on Graphics Hardware, EUROGRAPHICS 2005, 2005

[3] Daniel Weiskopf, Basic of GPU-Based Prog ramming, IEEE Visualization 2004 Tutorial, 2004

[4] Cyril Zeller, Introduction to the Hardware Graphics Pipeline, EUROGRAPHICS 2004 Tutorial, 2004

[5] Randy Fernando, Programming the GPU: High- Level Shading Languages, EUROGRAPHICS 2004 Tutorial, 2004

[6] Nvidia Developer, http://developer.nvidia.com/object/gpu_programming_guide.html

[7] Natalya Tatarchuk, Dynamic Parallax Occlusion Mapping with Approximate Soft Shadows, SIGGRAPH 2006, pp63-69, August 2006

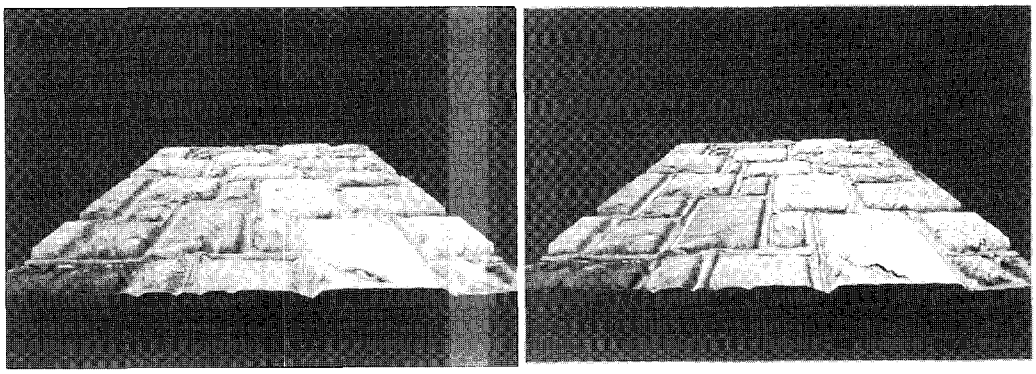
[8] Aaron Herzmann, Fast paint texture. In Second International Symposium on Non-Photorealistic Animation and Rendering (NPAR 2002), 9196.

<표 3> 디스플레이스먼트 매핑(버텍스 쉐이더)

```
void Dis_VertexShader(float4 position    : POSITION,
                      float3 normal      : NORMAL,
                      float2 texCoord    : TEXCOORD0,
                  out float4 oPosition   : POSITION,
                  out float2 oTexCoord   : TEXCOORD0,
                  out float3 oNormal     : TEXCOORD1,
                  out float3 lightDirection : TEXCOORD2,
                  out float3 halfAngle   : TEXCOORD3,
              uniform float  dis, // height scale factor
              uniform float3 lightPosition,  // Object-space
              uniform float3 eyePosition,    // Object-space
              uniform float4x4 modelViewProj,
              uniform sampler2D  decal,
              uniform sampler2D  heightMap)
{
// Displacement Mapping
  oPosition = mul(modelViewProj, position);
  oTexCoord = texCoord
// Calculate vertex normal
  float2 rCoord = float2(texCoord.x+0.01, texCoord.y);
  float2 aCoord = float2(texCoord.x, texCoord.y+0.01);
  float3 Hg = tex2D(heightMap, texCoord).xyz;
  float3 Ha = tex2D(heightMap, aCoord).xyz;
  float3 Hr = tex2D(heightMap, rCoord).xyz;
  float3 normM1 = float3(Hr.x-Hg.x, Ha.x-Hg.x, 1);
  float3 normM2 = float3(Hr.y-Hg.y, Ha.y-Hg.y, 1);
  float3 normM3 = float3(Hr.z-Hg.z, Ha.z-Hg.z, 1);
  float3 N = normalize(normM1+normM2+normM3 );
  oNormal = N;
  lightDirection = lightPosition - position.xyz;
  // Add the computation of a per-vertex half-angle vector
  float3 eyeDirection = eyePosition - position.xyz;
  halfAngle = normalize(normalize(lightDirection) +
                        normalize(eyeDirection));
}
```

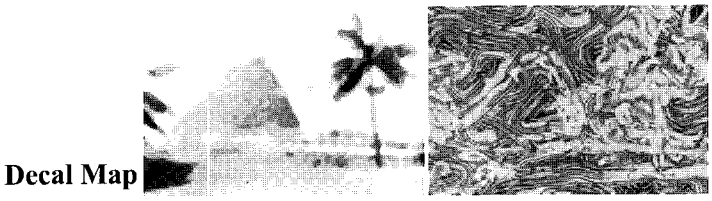
<표 4> 디스플레이스먼트 매핑(픽셀 쉐이더)

```
float3 expand(float3 v) { return (v-0.5)*2; }
void Dis_PixelShader(float2 decalTexCoord : TEXCOORD0,
                     float3 normal        : TEXCOORD1,
                     float3 lightDirection : TEXCOORD2,
                     float3 halfAngle     : TEXCOORD3,
                 out float4 color         : COLOR,
             uniform float  ambient,
             uniform float4 LMd, // Light-material diffuse
             uniform float4 LMs, // Light-material specular
             uniform float shininess, // Light-material shiness
             uniform float blend, // texture blending factor
             uniform sampler2D  decal,
             uniform sampler2D  heightMap,
             uniform samplerCUBE normalizeCube,
             uniform samplerCUBE normalizeCube2)
{
// Fetch and expand range-compressed normal
  float2 rCoord = float2(decalTexCoord.x+0.01, decalTexCoord.y);
  float2 aCoord = float2(decalTexCoord.x, decalTexCoord.y+0.01);
// Calculate Pixel Normal
  float3 Hg = tex2D(heightMap, decalTexCoord).xyz;
  float3 Ha = tex2D(heightMap, aCoord).xyz;
  float3 Hr = tex2D(heightMap, rCoord).xyz;
  float3 normM1 = float3(Hr.x-Hg.x, Ha.x-Hg.x, 1);
  float3 normM2 = float3(Hr.y-Hg.y, Ha.y-Hg.y, 1);
  float3 normM3 = float3(Hr.z-Hg.z, Ha.z-Hg.z, 1);
  float3 norm = normalize(normM1+normM2+normM3);
  float3 N = normalize(norm);
// Fetch and expand normalized light vector
  float3 normLightDirTex = texCUBE(normalizeCube,
                                   lightDirection).xyz;
  float3 normLightDir = expand(normLightDirTex);
// Fetch and expand normalized half-angle vector
  float3 normHalfAngleTex = texCUBE(normalizeCube2,
                                    halfAngle).xyz;
  float3 normHalfAngle = expand(normHalfAngleTex);
// Compute diffuse and specular lighting dot products
  float diffuse = saturate(dot(N, normLightDir));
  float specular = pow(saturate(dot(N, normHalfAngle)),shininess);
  if (diffuse <=0) specular=0;
  // Successive multiplies to raise specular to 8th power
  float3 decalTexture = tex2D(decal, decalTexCoord).xyz;
  float3 decalColor = LMd*(diffuse+ambient)*decalTexture;
  float3 NormalColor= LMs*specular;
  color.xyz= decalColor+NormalColor;
  color.w=1;
}
```

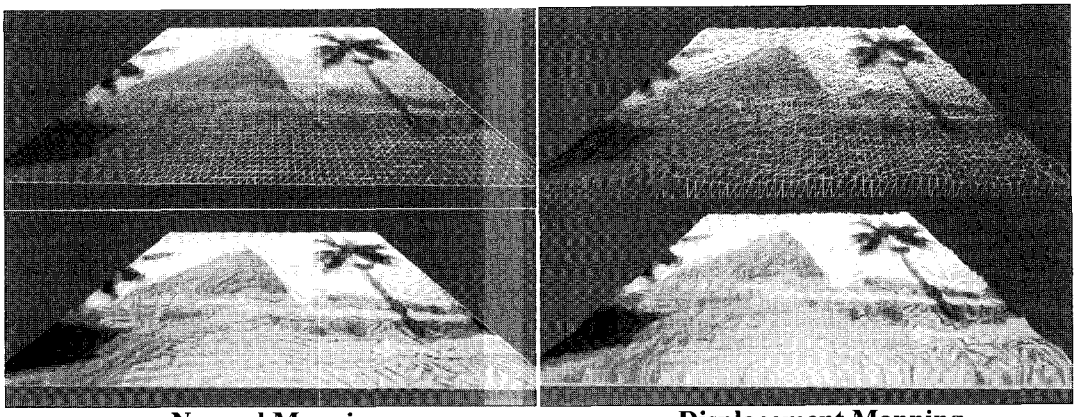


Vertex Normal　　Pixel Normal

<그림 8> 디스플레이스먼트 매핑(버텍스 노말과 픽셀 노말의 비교)



Decal Map　　**Heightfield Map**



Normal Mapping　　**Displacement Mapping**

<그림 9> 회화적 질감 표현