온라인 게임 서비스에서 캐릭터 외형 페인팅 시스템

김남훈* · 김상원*
*중앙대학교 한국게임사관학교

Character Skin Painting System for Online Games Service

Kim, Nam Hoon* · Kim, Sang Won*
*Dept. of Korea Game Academy., Chung-Ang University
E-mail : reilovekim@gmail.com, shabel@netsgo.com

요 약

온라인 게임 서비스를 위해 캐릭터 외형 페인팅를 하기 위해서는 3D 인터렉티브 텍스쳐 페인팅를 사용한다. 3D 인터렉티브 텍스쳐 페인팅은 3차원 공간상에서 3D 모델 데이터의 표면에 직접 브러쉬를 이용하여 페인팅 하는 것을 말한다. 3D 인터렉티브 텍스쳐 페인팅을 구현하기 위해서는 매개변수화와 브러쉬 드로잉이 중요하다. 본 논문에서는 3D 공간상에서 평균값좌표를 이용하여 3D 모델의 표면 메쉬상에 텍스쳐 페인팅을 수행하는 효과적인 방법을 제안한다. 기존 연구가 가지고 있는 문제점은 반복적인 매개변수화의 과정과 그로 인한 해상도의 큰 변화이다. 이러한 반복적인 매개변수화 과정을 피하기 위하여 주어진 메쉬에 한 번의 전역적 매개변수화를 적용시키고, 3차원상의 메쉬에 직접적으로 페인팅 할 수 있는 인터페이스를 제공하였다. 결과적으로 2차원으로의 텍스쳐 수정 과정 및 렌더링 과정을 반복적으로 적용할 필요가 없게 되어 3차원 모델러에게는 매우 효율적인 방법을 제공할 수 있다.

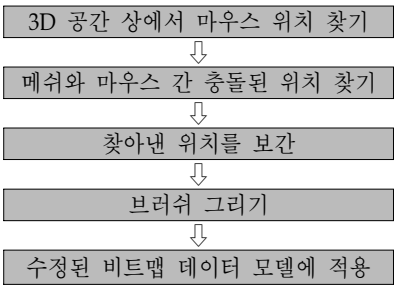
I. 서 론

컴퓨터 그래픽의 구현 방법에는 현실감 있는 가상세계를 만들기 위해서 3D 모델 데이터만으로 모든 것을 표현 하는 방법과 3D 모델 데이터 표면에 원하는 2D 이미지를 입히는 방법으로 표현 가능하다. 이 두 가지 방법을 배합해서 다양한 모델 데이터를 생동감 있게 만드는 방법도 있다. 일반적으로 컴퓨터 그래픽스를 구현하면서 모든 것을 3D 모델 데이터만으로 영화에 나오는 캐릭터와 같이 복잡한 3D 모델을 만드는 것은 시간이나 구현 속도 면에서 어려움이 많다. 그래서 텍스쳐라고 부르는 2D 비트맵 이미지를 이용한 텍스쳐 매핑 방법을 많이 사용한다. 텍스쳐 매핑은 3차원 공간에서 표면으로의 대응를 의미한다. 3D 모델 데이터의 표면에 2D 이미지를 입힘으로써 복잡한 3D 모델 데이터인 것처럼 보이도록 하여 현실감있게 표현하는 방법이다.

텍스쳐 페인팅은 3D 모델 데이터를 작업한 뒤 3D 모델 데이터의 표면에 2D 비트맵 이미지를 입히는 텍스쳐 매핑을 하기 위해 2D 비트맵 이미지를 만드는 작업을 텍스쳐 페인팅이라고 한다. 본 연구에서는 이러한 텍스쳐 페인팅을 3D 모델 데이터를 불러들인 3D 뷰화면에서 직접 작업을 수행하기 때문에 3D 인터렉티브 텍스쳐 페인팅이라고 한다. 3D 인터렉티브 텍스쳐 페인팅은 Takeo Igarashi et. al [1], Olga Sorkine [2], Pat Hanrahan et. al [3]에 의해 많이 연구되어져왔다. 특히 Takeo Igarashi의 연구에서는 3D 모델 데이터의 표면에 브러쉬로 드로잉 되어 진 부분만 매개변수화하여 결과물을 나타내고 그로인하여, 이미지의 해상도가 낮아지는 문제점이 발생하였다.

II. 3D 인터렉티브 텍스쳐 페인팅 시스템

3D 인터렉티브 텍스쳐 페인팅의 시스템은 크게 세 부분으로 구성되어있다. 3D 상에서 마우스의 위치를 찾기 위한 광선 계산과 자연스러운 브러쉬를 그리기 위한 보간법[4], 보간법에 의해 찾아낸 점들을 따라가며 브러쉬 그리기로 구성되어있다.

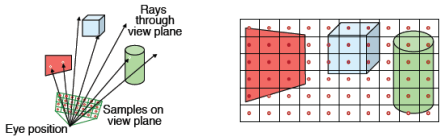


<그림 1> 시스템 구조

광선 계산은 3D 상에서 마우스의 위치를 3D 모델 데이터의 메쉬와 관련하여 찾게 된다. 이렇게 찾아낸 위치는 평균값좌표를 이용하여 메쉬 상에서 어디에 위치하는지 찾게 된다. 자연스러운 브러쉬를 그리기 위해서는 각 점과 점을 보간법을 이용하여 그려야만 한다. 보간법은 단순히 직선으로 연결할 수도 있지만, 곡선으로도 연결을 할 수 있다. 보간법으로 찾아낸 브러쉬 경로를 이용하여 부드러운 페인팅이 가능하게 된다. 그림 1은 3D 인터렉티브 텍스쳐 페인팅 시스템 구조를 보여주고 있다.

2.1 광선 계산

레이 캐스팅(Ray Casting)은 카메라의 시점에서부터 특정 픽셀을 통해 빛을 투사해서 이 빛이 물체에 닿거나 보이는 영역에서 벗어날 때까지 따라감으로써 해당 픽셀의 색을 결정해 렌더링 이미지를 생성하는 방법이다. 그 빛이 아무런 물체에도 닿지 않는다면 해당 픽셀은 검은색이 되고, 빛이 어떤 오브젝트 표면에 도달하게 되면 그 빛이 충돌된 지점에서의 오브젝트 색상을 계산해서 그 값을 해당 픽셀의 색으로 결정한다. 전체 장면에서 생성될 전체 이미지의 각각의 픽셀마다 순차적으로 위와 같은 과정을 반복한다. (그림 2).



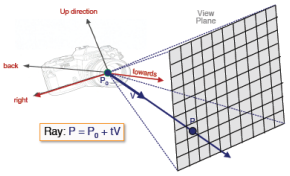
(a) (b)

<그림 2> 레이 캐스팅[5]

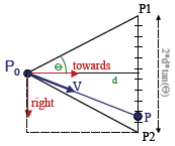
레이 캐스팅은 작업 공간에 존재하는 오브젝트가 여러 개 일 경우 그 계산방식의 특성상 각각 따로 존재하는 것처럼 렌더링하기 때문에 물체간의 반사나 투명도가 있는 물체를 표현하거나 물체 서로 간에 그림자를 드리우는 효과를 계산할 수가 없다. 따라서 그러한 효과를 흉내내기 위해서 일종의 편법들을 사용하게 되는데 예를 들어 표면 반사를 위해서는 반사 매핑(Reflection mapping)을 사용하고, 투명 오브젝트를 위해서는 겹쳐지는 두 오브젝트의 색상을 섞어서 사용하며, 그림자는 Z-buffer의 정보를 적절히 활용해서 만들어 주게 된다. 본 논문에서는 레이 캐스팅 방법을 이용하여 3D 모델 데이터의 표면의 메쉬와 광선 간의 충돌을 계산하는데 사용한다[5].

2.2 광선과 메쉬 다각형 간의 충돌

광선과 메쉬 다각형 간의 충돌 위치를 찾기 위해서는 우선 픽셀을 통과하는 광선을 설정해야한다.



<그림 3>
픽셀을 통과하는 광선[5]



<그림 4>
뷰스펙트럼[5]

그림 3에서 광선의 방정식은 $P = P_0 + tV$ 이다. 그림 4에서 θ은 프러스텀(frustum half-angle) 각의 반이고, d는 뷰 평면(view plane)이다. 그리고 right = towards × up 이다. 여기서 $P1$, $P2$는

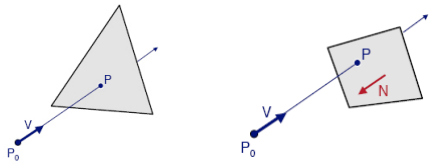
$$P1 = P_0 + dt - d\tan(\theta)r$$
$$P2 = P_0 + dt - d\tan(\theta)r$$

(t=towards, r=right)

이고, P와 V는 다음과 같다.

$$P = P1 + (i+0.5)\frac{(P2-P1)}{width}$$
$$= P1 + (i+0.5)\frac{2d\tan(\theta)r}{width}$$
$$V = \frac{(P-P_0)}{\| P-P_0 \|}$$

광선과 삼각 메쉬 간의 충돌은 우선 평면과 광선 간의 충돌을 확인해야한다. 그런 다음 삼각 메쉬 안에 충돌 점이 있는지 확인한다. (그림 5 (a)).



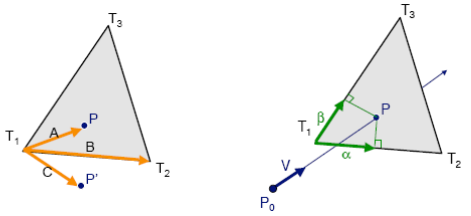
(a) (b)

<그림 5> 광선과 평면의 충돌[5]

그림 5 (b)에서 광선의 방정식은 $P = P_0 + tV$ 이고, 평면의 방정식은 $P \cdot N + d = 0$ 이다. 여기서 P에 대입하면, $(P_0 + tV) \cdot N + d = 0$이 된다. 이 식으로부터 해법을 구할 수 있다.

$$t = -(P_0 \cdot N + d)/(V \cdot N)$$
$$P = P_0 + tV$$

평면과의 계산을 한 다음에는 메쉬 안에 광선이 존재하는지 체크해야한다.



(a) (b)

<그림 6> 메쉬 안에 광선이 있는지 확인[5]

그림 6 (a)의 $A \times B$는 $C \times B$로부터 마주보고 있는 방향에서 충돌되는 점을 것이다.

그림 6 (b)에서 α, β를 계산하면,

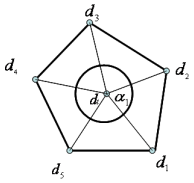
$$P = \alpha(T_2 - T_1) + \beta(T_3 - T_1)$$ 이고

만약 삼각 메쉬 안에 충돌되는 점이 있다면

$$0 \le \alpha \le 1,\ 0 \le \beta \le 1$$
$$\alpha + \beta \le 1$$ 이 된다.

2.3 평균값좌표

2차원 삼각 메쉬에 있는 임의의 정점은 그의 이웃하는 정점들의 볼록 조합으로 표현이 가능하다. [6][7]



<그림 7> 이웃 정점에 의한 내각 및 평균값 좌표

$d_1, d_2, \cdots, d_m$가 그림 7에서 보이는 것처럼 정점 d_i에 대해서 반 시계 방향으로 있는 이웃하는 점들이라 하자. 원은 반지름이 1이며, 이웃하는 정점들 d_j와 d_{j+1}가 정점 d_i와 이루는 내각 $a_j = \measuredangle d_j d_i d_{j+1}$은 예각을 이루고 있으며, 이를 바탕으로 다음과 같이 평균값 좌표 ω_j와 가중치 λ_j를 정의하면

$$\omega_j = \frac{\tan(\alpha_{j-1}/2) + \tan(\alpha_j/2)}{\| d_j - d_i \|},$$
$$\lambda_j = \frac{\omega_j}{\sum_{k=1}^{m} \omega_k}$$

가중치 λ_j는 다음과 같은 볼록 조합 조건을 만족하게 된다.

$$d_i = \sum_{j=1}^{m} \lambda_j d_j, \qquad \sum_{j=1}^{m} \lambda_j = 1,\ \lambda_j \geq 0$$

위의 식을 유도하기 위하여 정점 d_i를 중심으로 하는 극좌표계를 이용하여 이웃하는 정점들을 표시한다.

$$d_j = d_i + r_j(\cos\theta_j,\ \sin\theta_j),$$
$$r_j = \| d_i - d_j \|,$$
$$\theta_j = Aug(\overrightarrow{d_i d_j})$$

그러면

$$\frac{d_j - d_i}{\| d_j - d_i \|} = (\cos\theta_j,\ \sin\theta_j),\ \alpha_j = \theta_{j+1} - \theta_j$$

일반적으로 원주 상에 놓여 있는 점들의 법선 벡터의 합은 0 벡터라는 사실이 알려져 있다. 이를 이용하여 다음의 식을 유도한다.

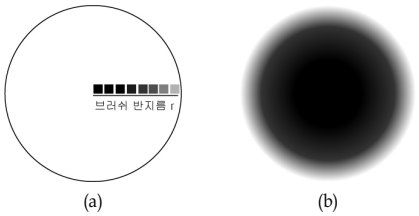
$$\begin{aligned}
0 &= \int_0^{2\pi} (\cos\theta, \sin\theta)\, d\theta \\
&= \sum_{j=1}^{m} \int_{\theta_j}^{\theta_{j+1}} (\cos\theta, \sin\theta)\, d\theta \\
&= \sum_{j=1}^{m} \int_{\theta_j}^{\theta_{j+1}} \left\{ \frac{\sin(\theta_{j+1} - \theta)}{\sin\alpha_j} (\cos\theta_j, \sin\theta_j) + \frac{\sin(\theta - \theta_j)}{\sin\alpha_j} (\cos\theta_{j+1}, \sin\theta_{j+1}) \right\} d\theta \\
&= \sum_{j=1}^{m} (\cos\theta_j, \sin\theta_j) \frac{1}{\sin\alpha_j} \int_{\theta_j}^{\theta_{j+1}} \sin(\theta_{j+1} - \theta)\, d\theta \\
&\quad + \sum_{j=1}^{m} (\cos\theta_{j+1}, \sin\theta_{j+1}) \frac{1}{\sin\alpha_j} \int_{\theta_j}^{\theta_{j+1}} \sin(\theta - \theta_j)\, d\theta \\
&= \sum_{j=1}^{m} (\cos\theta_j, \sin\theta_j) \frac{1 - \cos\alpha_j}{\sin\alpha_j} + \sum_{j=1}^{m} (\cos\theta_{j+1}, \sin\theta_{j+1}) \frac{1 - \cos\alpha_j}{\sin\alpha_j} \\
&= \sum_{j=1}^{m} (\cos\theta_j, \sin\theta_j) \left\{ \frac{1 - \cos\alpha_j}{\sin\alpha_j} + \frac{1 - \cos\alpha_{j-1}}{\sin\alpha_{j-1}} \right\} \\
&= \sum_{j=1}^{m} (\cos\theta_j, \sin\theta_j) \{ \tan(\alpha_j/2) + \tan(\alpha_{j-1}/2) \} \\
&= \sum_{j=1}^{m} \frac{d_j - d_i}{\| d_j - d_i \|} \{ \tan(\alpha_j/2) + \tan(\alpha_{j-1}/2) \} \\
&= \sum_{j=1}^{m} \frac{\tan(\alpha_j/2) + \tan(\alpha_{j-1}/2)}{\| d_j - d_i \|} (d_j - d_i)
\end{aligned}$$

그러므로 λ_j가 볼록 조합 조건을 만족함을 알 수 있다.

$$\sum_{j=1}^{m} \omega_j d_i = \sum_{j=1}^{m} \omega_j d_j,$$
$$\omega_j = \frac{\tan(\alpha_{j-1}/2) + \tan(\alpha_j/2)}{\| d_i - d_j \|}$$
$$(\sum_{j=1}^{m} \omega_j) d_i = \sum_{j=1}^{m} \omega_j d_j,$$
$$d_i = \sum_{j=1}^{m} \frac{\omega_j}{\sum_{j=1}^{m} \omega_j} d_j$$

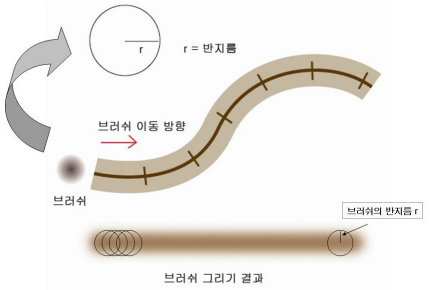
2.4 브러쉬 그리기

브러쉬가 지난간 경로를 보간법에 의해서 찾게 되면, 해당 경로를 따라서 브러쉬를 그려주기만 하면 된다. 브러쉬는 브러쉬의 형태가 원일 경우, 원의 중심에 가까워질수록 짙은 명도를 가지고 멀어질수록 옅은 명도를 가진다. (그림 8, 그림 9).



<그림 8> 브러쉬 반지름에 따른 투명도

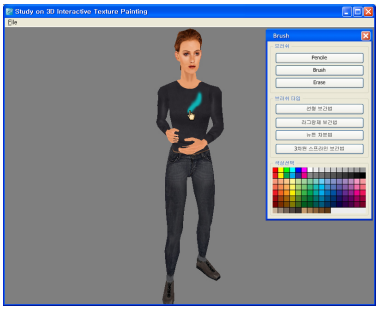
일정한 반지름의 길이만큼은 투명도가 거의 값을 가지지 않으며, 반지름이 조금씩 길어 지면서 투명도 값도 순차적으로 커지게 된다. 그림 8 (a)는 반지름에 따른 투명도의 예를 단적으로 보여주고 있으며, 이에 따른 결과는 그림 8 (b)에 나타내고 있다. 그림 9에서는 이렇게 계산된 브러쉬가 정해진 경로를 따라서 그려지는 것을 표현한 것인데, 여기서 중요한 것은 정해진 경로가 등간격으로 나누어져 있고, 브러쉬 형태의 오브젝트는 이 정해진 경로의 곡률에 따라서 그려져야 한다는 것이다.



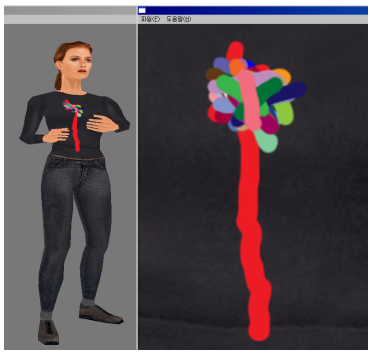
<그림 9> 브러쉬 경로 따라 그리기

III. 실험 결과 및 분석

실험을 위해서 Intel Core2 Quad 2.6GHz, 2GB 램, ATI Radeon HD3450 그래픽카드, IBM 호환 PC와 Intel Pentium M Processor 1.8 GHz, 1GB 램, ATI Mobility Radeon X700 그래픽카드, 노트북에서 수행하였다. 3D 라이브러리는 DirectX 9 (August 2008) 버전을 사용하여 Visual Studio 2008에서 C++를 사용하여 구현하였다. 3D 모델 데이터의 메쉬는 게임 개발에 일반적으로 많이 사용되는 1526개(그림 10)(그림 11)를 사용하였다.

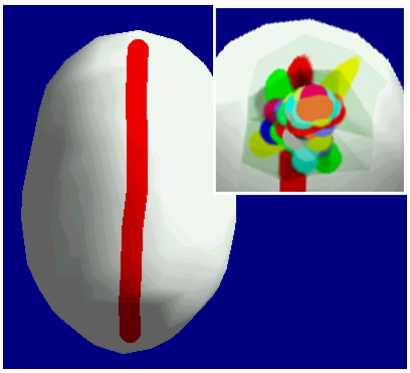


<그림 10> 아바 캐릭터 결과 이미지



<그림 11> 텍스쳐 맵핑 소스 결과 이미지

그림 10과 그림 11은 최종적으로 브러쉬 페인팅을 적용한 결과를 보여주고 있다. 브러쉬 그리기가 보간법에 의해서 잘 그려지는 것을 알 수 있다. 그림 11에 사용된 텍스쳐 맵핑 소스의 크기는 1024 × 1024 픽셀을 사용하였으며 30번의 브러쉬 그리기를 수행했음에도 불구하고 이미지의 손실이 없음을 알 수 있다.



<그림 12> Takeo Igarashi 텍스쳐 맵핑 결과

그림 12는 빨간색 브러쉬를 이용하여 3D 모델에 그리기를 수행한 그림과 이러한 그리기를 30번 반복 수행한 결과를 우상단에 보여주고 있다. Takeo Igarashi의 연구에서 문제가 되는 텍스쳐 맵핑 소스의 퀄리티가 많이 낮아지는 것을 알 수 있으며, 또한 각각의 이미지의 퀄리티가 다르게 나타나는 현상도 발생한다. 이러한 결과는 브러쉬를 그릴 때 마다 반복적인 매개변수화로 인하여 이미지의 이동 및 변환이 반복되기 때문이다.

IV. 결 론

본 논문에서는 레이 캐스팅를 이용하여 3D 공간상에서 마우스의 위치를 찾아내어 3D 모델 데이터와의 충돌 유무를 확인하였다. 또한 메쉬와 마우스 간의 충돌 되는 지점을 찾기 위해서 평균값좌표를 이용하였다. 이것을 이용하여 브러쉬의 경로를 알 수 있었고, 부드러운 브러쉬 그리기를 위하여 보간법을 이용하였다. 본 연구에서는 한 번의 전역적 매개변수화를 수행한다. 결과적으로 2차원으로의 텍스쳐 수정 과정 및 렌더링 과정을 반복적으로 적용할 필요가 없게 되어 이미지의 퀄리티를 처음과 같이 유지 할 수 있으며, 실행 속도 또한 빠르다. 제안된 3D 인터렉티브 텍스쳐 페인팅 방법의 완성도를 높이기 위해서는 사람의 눈이나, 천의 질감 등 세밀한 작업이 가능하도록 하는 연구가 필요하며, 페인팅된 이미지가 메모리상에서 실시간으로 3D 모델 데이터로 빠르게 매핑되는 방법에 대한 연구도 필요하다.

참고문헌

[1] Takeo Igarashi and Dennis Cosgrove, Adaptive Unwrapping for Interactive Texture Painting, ACM Symposium on Interactive 3D Graphics, ACM I3D'01 Research Triangle Park, NC, March 19-21, 2001, pp.209~216.

[2] Olga Sorkine and Daniel Cohen-Or and Rony Goldenthal and Dani Lischinski, "Bounded-distortion Piecewise Mesh Parameterization", IEEE Visualization, 2002, pp. 220~234.

[3] Pat Hanrahan and Paul E. Haeberli, "Direct WYSIWYG painting and texturing on 3D shapes", In Proceedings of SIGGRAPH 90, August. 1990, pp.215~223.

[4] 지영준, 김화준, 허정권, "C로 구현한 수치해석", 높이깊이, 2002 pp.257~283

[5] Greg Humphreys, "Intro Graphics", Lecture Note, 2004.

[6] Hyoungseok Kim, "Mesh Parameterization based on Mean value coordinates", preprint, 2008.

[7] Michael S. Floater, "Mean Value Coordinates" Comp. Adied Geom, Design 20, 2003, pp.19~27.