실시간 Fur 시뮬레이션 렌더링

Realtime Fur Simulation Rendering

김동겸, DongKyoum Kim*, 김지인, Jee-In Kim**, 김형석, HyungSeok Kim***
건국대학교 인터넷 미디어 공학부

요약 실시간 렌더링에서 Fur는 모피등과 같이 매우 복잡한 표면을 표현하는 문제로 가상세계의 사실감을 높이는데 매우 중요한 요소이다. 복잡한 Fur의 실시간 렌더링을 위하여 다수의 방법이 제안되어 왔으나, Fur를 사실처럼 보여지게 하는 측면에서, 기존의 정적인 표현으로서는 한계점이 존재한다. 본 논문에서는 중력 및 외력에 의한 시뮬레이션을 통한 Fur의 실시간 Animation 방법을 제안한다. 기본 구조는 모피의 볼륨을 구성하는 n개의 Shell과 Shell의 표현을 보강하는 Fin의 구조로 이루어져 있고, Shell과 Fin의 공유 Vertex 배열을 통해 이 두 가지 구조를 하나로 통합한다. 이 공유 Vertex 배열에 본 논문에서 제안하는 중력 및 외력에 의한 시뮬레이션을 적용하여 공유 Vertex 배열을 변형시킨다. 이 후 변형된 공유 Vertex배열을 기반으로 Rendering을 수행하게 된다. 본 논문에서 제안하는 방법을 사용하여, 정적인 Fur Rendering이 아닌 동적으로 움직이는 Fur Rendering을 사용 함 으로써 좀 더 높은 현실감을 느낄 수 있을 것으로 기대한다.

핵심어: Fur Simulation, Fur Rendering, Realtime Simulation

본 연구는 서울시 산학연 협력사업(10581)의 지원에 의하여 연구되었음.
*주저자 : 건국대학교 인터넷미디어공학부 학사과정 김동겸 e-mail: fannori@naver.com
**공동저자 : 건국대학교 인터넷미디어공학부 교수 김지인 e-mail: jnkm@konkuk.ac.kr
***교신저자 : 건국대학교 인터넷미디어공학부 교수 김형석 e-mail: hyuskim@konkuk.ac.kr

1. 서론

하드웨어의 급속한 발전에도 불구하고, 물리 역학에 기반한 Fur를 실시간으로 렌더링 하기 위해서는 아직 어려운 점이 많다. 하지만 사용자들은 좀 더 실감나는 영상을 원하고 있기 때문에 이를 위해 다양한 기법[1,2,3,4]을 통해 실시간으로 처리하려는 연구가 진행되고 있다.

Lengyel등은 Shell과 Fin의 구조를 사용하여 Fur를 표현함으로써 실시간 렌더링을 구현하였다[1]. 또한 Lapped Texture기법[6]을 사용하여 Shell 텍스쳐 좌표를 계산하여 Shell Texture 비용을 크게 절감할 수 있음을 보였다[1]. Yang 등은 시점과 Shell 평면의 각도를 계산하여 그 각도에 따라 Shell Layer의 개수를 동적으로 조절하여 속도를 크게 개선하였다[4].

이와 함께 좀 더 사실적인 표현을 위하여도 다수의 연구가 수행되어 왔다. Dave등은 사자의 갈기를 표현하기 위해 Mass Spring 구조를 사용하여 부드러운 Fur를 표현하였다.[7] Meyer등은 volume 텍스쳐를 애니메이션 시킬 때, surface deformation, mapping deformation, texel content modification등의 방법을 통해 volume 텍스쳐를 효과적으로 변형 시키는 방법을 제안하였다.[8]

Lengyel등의 방법[1]에서 Shell이란 Fur의 단면정보를 텍스쳐로 저장하고 있는 구조로서 여러 개의 Shell 구조를 겹침으로써 Fur의 전체적인 볼륨정보를 표현해 준다. Fin이란 Shell만을 사용해서 표현할 때 일정각도에서 텍스쳐가 보이지 않는 약점을 보완하기 위해 사용되는 구조로서 Shell이 가지고 있지 않은 수직 정보를 텍스쳐로 저장하고 있다. 이러한 구조는 Volume 텍스쳐나 혹은 각각의 strand를 별도로 표현하지 않는 경우 실시간 렌더링을 위하여 가장 적합한 구조로 사용되어 오고 있다.

본 논문은 Shell과 Fin 구조를 기반으로 하며, 사실감 있는 실시간 애니메이션을 구축하였다. 사용자가 렌더링된 Fur를 보았을 때 애니메이션이 없이 정적인 형상만을 보는 경우 실제 가상세계에서 그 사실감은 매우 제한적이게 된다. 이에 대한 해결책으로서 본 연구에서는 중력 및 외력 시뮬레이션을 기반으로 하는 애니메이션을 표현한다.

본 논문에서 제안하는 시스템은 기존의 연구에서의 정적인 Fur를 표현해 주는 것뿐만이 아니라, 실시간 시뮬레이션을 기반으로 한 애니메이션을 통해 동적인 Fur를 표현해 줌으로써 사용자에게 표현된 객체의 현실감을 증대시키는 효과를 가져온다.

2. Background

본 절에서는 Lengyel등의 방법을 기반으로 본 연구에서 사용하는 Shell 및 Fin 구조를 간략히 기술한다.[1]

Shell이란 투명도를 가지는 텍스쳐 여러 장을 일정 간격을 두고 겹쳐서 Fur의 전체적인 볼륨을 표현해주는 구조이다. 겹쳐진 여러 장의 텍스쳐를 수직에 가까운 위치에서 바라보았을 때, 이들이 하나의 연속된 개체인 듯한 일종의 착시효과를 기대할 수 있게 된다[그림 2-1,2]. 하지만 이 Shell 텍스쳐를 [그림 2-3]과 같이 수평에 가까운 각도에서 바라보게 되면, Shell 텍스쳐로는 해결 할 수 없는 공간이 발생한다. 이런 문제를 해결하기 위해 Shell Layer 사이의 빈 공간을 채워줄 Fin 구조를 사용하게 된다.

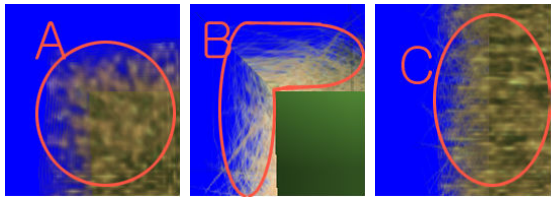

그림 1. Direct-X SDK 9.0 "Fur" sample[4]
(A)Shell, (B)Fin, (C)Shell+Fin

그림 2-1. Shell 텍스쳐와 View 좌표계

그림 2-2. 수직에 가까운 위치에서 바라본 Shell 텍스쳐

그림 2-3. 수평에 가까운 위치에서 바라본 Shell 텍스쳐

Fin 구조는 Shell구조가 표현해 줄 수 없는 수평 시야각에서의 수직 정보를 표현해 주는 구조이다. Fin 구조는 단순히 Shell구조의 단점인 Shell Layer 사이의 빈 공간을 채워주는 것뿐만 아니라 Fur자체를 좀 더 풍성하게 표현해 준다. 또한 Shell 텍스쳐 보다 디테일한 텍스쳐 정보를 표현해 줌으로써 Fur 자체를 좀 더 디테일하게 표현해 주게 된다.

그림 3-1. Shell

그림 3-2. Shell + Fin

3. 시스템 개요

3.1 기본 구조

본 논문에서는 Shell과 Fin의 구조를 기반으로 한다. 그러나, 일반적인 Fin을 사용하는 경우 Fin의 단순함으로 인하여 부드러운 애니메이션이 어려운 단점을 갖는다. 또한 Fin과 Shell 구조가 독립적으로 존재하여 동적인 구조에 있어 불연속 점이 발생할 여지가 많은 단점이 있다.

본 연구에서는 이를 개선하기 위하여, Fin과 Shell의 구조를 하나로 통합한 구조를 제안한다.

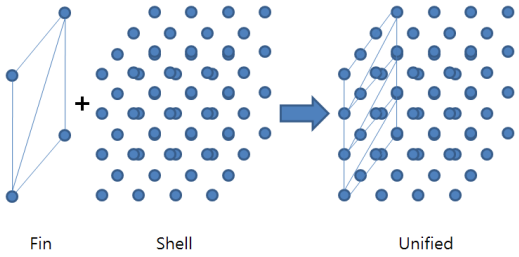

그림 4. 통합된 Fin과 Shell

두 구조를 하나로 통합하게 되면, 기본적으로 Vertex정보를 하나로 통합하게 됨으로써 전체 Vertex정보가 축소되고, 시뮬레이션 과정에서 공유된 Vertex 정보를 사용함으로써 Fin과 Shell이 함께 유기적으로 동작하게 되며, Fin을 표현해주는 Vertex의 개수가 늘어남에 따라 Fin을 좀 더 유연하게 표현할 수 있게 된다[그림 5-1].(그림에서는 효과를 나타내기 위해서 임의의 직선 텍스쳐를 사용했다.)

그림 5-1. 32개의 Face를 갖는 Fin의 시뮬레이션

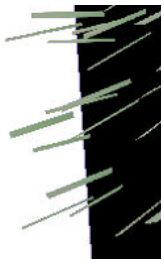

그림 5-2. 2개의 Face를 갖는 Fin의 시뮬레이션

3.1.1 Fin과 Shell의 생성

Shell과 Fin의 Mesh 데이터는 Base Mesh의 데이터에 기반하여 생성한다. Shell과 Fin의 공유되는 Vertex는 다음과 같은 간단한 수식을 통해 구할 수 있다.

$$new\_v[I] = BM.v[I] + (BM.n[I]*SL\_D*S\_I) \quad (1)$$

- $BM.v$ = Base Mesh.vertex array
- $BM.n$ = Base Mesh.normal_vec_array
- SL_D = Shell Layer Distance
- S_I = Shell Layer Index
- I = Base Mesh v_index $i+(S\_I*BM.num\_vertex)$

새로운 Fin과 Shell의 Vertex(new_v)는 생성될 기반이 되는 Base Mesh의 Vertex(BM.v)위치로부터 Base Mesh의 Normal Vector(BM.n)방향으로 Shell Layer간 거리(SL_D)만큼 떨어진 곳에 위치하게 된다. Shell Layer가 증가 될수록 Base Mesh로부터 멀리 떨어지지만 Shell Layer간의 거리는 유지하게 된다. 수식 (1)에서의 인덱스 I는 n개의 각 Shell Layer에서도 Base Mesh의 Vertex Index값을 유지하기 위해 Base Mesh의 Vertex Index값에 Base Mesh의 총 Vertex 개수 * Shell Index 값을 더했다. 이를 통해 한 Base Mesh Vertex에 관련된 모든 Shell의 Vertex를 접근할 수 있다.

수식(1)을 통해 Shell 과 Fin의 Vertex를 계산하는 과정에 있어서 Base Mesh의 Normal Vector(수식 (1)의 BM.n)를 수정하여 Shell 과 Fin의 Vertex 위치를 바꿈으로써 Fur의 결을 임의로 생성하는 효과를 낼 수 있다. 이는 그림[2-1]에 볼 수 있듯이 Shell 텍스쳐를 어느 방향에서 바라보는가 에 따라 Fur가 보이는 것이 틀려 보이기 때문에, Vertex의 위치를 바꿈으로 해서 보여 지는 텍스쳐의 위치도 바뀌게 되고, 결국 Fur에 결이 생성되는 효과를 생성할 수 있게 된다.

그림 6-1. 노말 수정 전

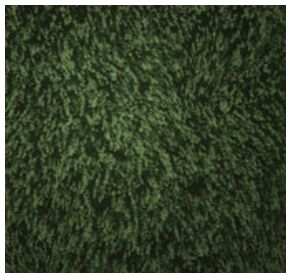

그림 6-2. 노말 수정 후

3.1.2 Shell의 텍스쳐 생성

본 논문에서의 Shell 텍스쳐는 Random값에 의해 Seed값이 정해지고 그 Seed값을 토대로 텍스쳐를 생성한다. 초기 Seed값의 RGB값을 XYZ값으로 치환하여 텍스쳐가 생성될 Normal Vector 데이터로 활용한다.

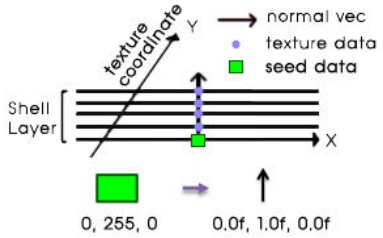

그림 7. RGB -〉 XYZ

Seed 값은 사용자가 설정한 빈도수와 임의의 Random값에 의해 채워지게 된다. 이때 주의할 점은 Seed값을 Vector로 활용하기 위해 Y축 성분인 Green 값을 높은 값으로 설정해야 한다는 것이다. Seed값이 정해진 뒤 이 값을 Vector 단위로 환산하여 각 Shell Layer의 텍스쳐 생성에 사용한다. 텍스쳐 데이터는 2차원 데이터이고 Shell Layer의 개수만큼의 텍스쳐가 있기 때문에 Shell Layer를 Y축 성분으로 하는 3차원으로 생각할 수 있게 된다. 텍스쳐 데이터 생성은 Seed Vector의 y값이 각 Shell Layer의 위치와 같아질 때의 x,z성분을 int단위로 변환하여 텍스쳐의 x,y성분으로 좌표를 설정하여 데이터를 생성할 수 있다.

3.2 시뮬레이션

본 연구에서의 시뮬레이션은 Fur의 애니메이션을 위하여 중력에 대한 시뮬레이션과 함께 다른 여러 시뮬레이션 데이터를 선형보간하여 처리한다. 여기에서 고려하는 다른 시뮬레이션 데이터에는 외력에 의한 시뮬레이션이 있으며, 밀도에 의한 시뮬레이션 등을 추가할 수 있다.

3.2.1 중력 시뮬레이션

본 논문에서의 중력 시뮬레이션은 간단한 수식을 통해 각 점들을 새로운 위치로 이동시킨다. 회전률에 사용하는 수식은 다음과 같다.

$$theta = dot(V,\ V2) * RN / stability \qquad (2)$$

수식에서의 stability는 모피가 자신의 원래 외형을 유지하려는 힘(복원력)에 관련된 변수이다. RN은 현재 노드(=Vertex)에서 Growing Vector방향으로 남은 노드의 개수이며 이는 현재 노드가 지탱해야할 나머지 노드에 대한 무게를 의미한다.

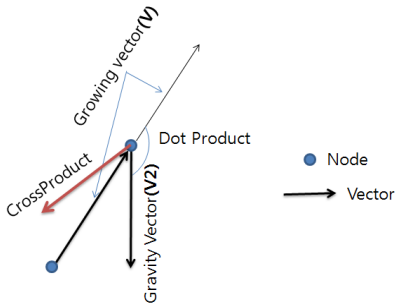

그림 8. 회전각 구하기

V와 V2의 외적과 수식을 통해 나온 회전 값들을 토대로 각 점들을 회전시키게 된다. 현재 노드에서의 회전 값들을 이후의 노드들에 모두 적용하여 회전시키게 된다. 이 후의 노드들도 같은 과정을 반복하여 회전 변환을 하게 된다.

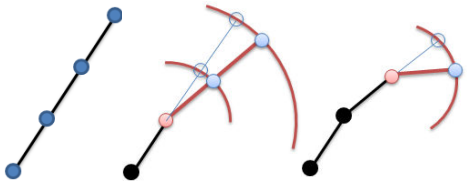

그림 9. 노드(Vertex) 회전 변환

모든 노드에 대해 이런 Deformation을 수행하게 되면 중력에 대한 간단한 시뮬레이션이 끝나게 된다. 이후로 변환된 Vertex를 토대로 렌더링을 하게 된다.

3.2.2 외력 시뮬레이션

중력 이외의 다른 외력에 의한 움직임을 표현하기 위해 Base Mesh의 각 Vertex마다 받는 힘의 크기를 저장하는 Force Field를 만들었다[그림 10]. Force Field의 각 Vertex에 할당된 Force Vector는 그 Vertex와 연결되어 있는 Shell과 Fin의 공유 Vertex들에게 영향을 미친다.

외력에 의한 시뮬레이션은 중력에 의한 것과는 다르게 보다 더 단순한 방법으로 계산된다. 시뮬레이션의 빈도수를 고려했을 때, 중력에 의한 시뮬레이션 보다 외력에 의한 시뮬레이션의 수행이 월등히 많으므로 많이 단순화될 필요가 있었다.

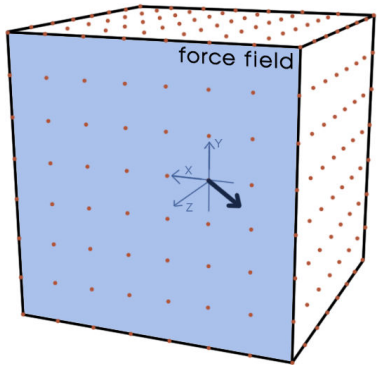

그림 10. Force Field

그래서 외력에 의한 시뮬레이션은 모든 노드에 대한 반복 회전[그림 9]이 아닌, 연관된 모든 노드를 같은 각도로 한번만 회전해 주게 된다[그림 11]. 각 Vertex에서의 Force Vector와 Normal Vector의 내적과 외적을 통해 회전각과 회전축을 구할 수 있다.

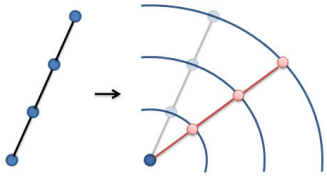

그림 11. 외력 시뮬레이션의
회전 변환

3.3 애니메이션

본 논문에서는 연산비용의 절약을 위해 매 Render 마다 시뮬레이션을 수행하는 것이 아니라, 시뮬레이션을 수행한 결과를 선형보간 하게 된다. 본 논문에서의 애니메이션을 위해 필요한 데이터는 Shell과 Fin의 공유 Vertex와 같은 크기의 애니메이션 데이터 배열 두 개를 필요로 한다. 하나의 배열에는 중력에 의해 Fur가 운동한 후 최종적으로 위치해야할 위치정보를 저장하고 있다([그림 12]의 Animation Data 1). 또 하나의 배열에는 외력에 의해 변형된 위치를 저장하고 있다([그림 12]의 Animation Data 2). 즉, 하나의 배열에는 시작 위치를 다른 하나의 배열에는 최종 위치를 저장한다. 이 두 가지 배열을 선형보간 하여 Fur의 움직임을 표현해 준다.

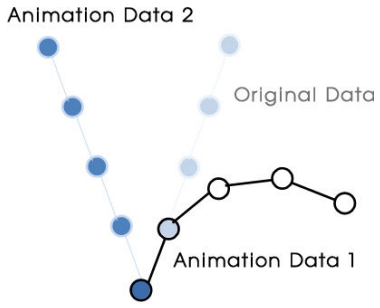

그림 12. 보간 될 애니메이션 데이터

4. 결과

본 논문에서 제안한 방법의 결과는 다음 그림과 같다. 3D 모델은 정사각형 Mesh와 Teapot Mesh를 사용했다. Shell Layer의 개수는 16개이고 텍스쳐 사이즈는 256*256이다.

CPU : Core2Duo E6400 2.13GHz

VGA : NVIDIA GeForce 7600GT (256MB)

RAM : 2G

표 1. Mesh 1 "Box.obj"

	Base	Shell	Fin	Sum
Vertex	296	4736	4736	5032
Face	588	9408	26460	36456
FPS				≒ 30

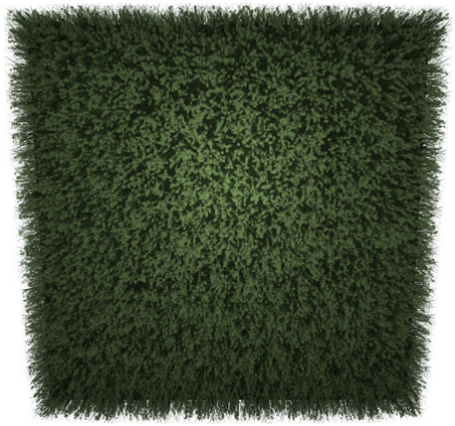

그림 13. 시뮬레이션 전

그림 14. 시뮬레이션 후

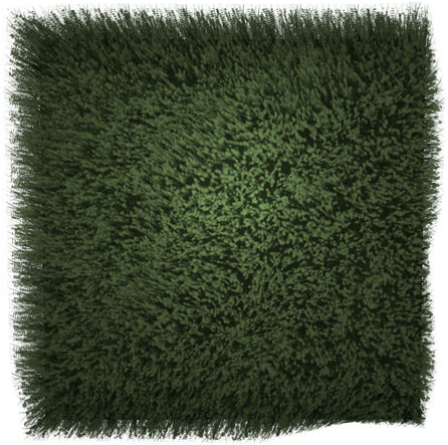

그림 15. 외력(바람) 시뮬레이션 1

그림 16. 외력(바람) 시뮬레이션 2

표 2. Mesh 2 "Teapot.obj"

	Base	Shell	Fin	Sum
Vertex	302	4832	4832	5134
Face	552	8832	25560	34944
FPS				≒ 30

그림 17-1. teapot 밀도 15

그림 17-2. teapot 밀도 30

표 3. Mesh 3 "Teapot2.obj"

	Base	Shell	Fin	Sum
Vertex	882	13152	13152	14034
Face	1560	24960	71400	97920
FPS				≒ 10

그림 18-1. teapot 밀도 15

그림 18-2. teapot 밀도 30

5. 결론 및 향후 연구

본 논문에서는 Shell과 Fin의 통합된 구조를 통해 제안한 중력 및 외력에 의한 시뮬레이션을 렌더링 하였다. 선형보간을 통한 애니메이션을 수행함으로서 Fur에 대한 현실감을 증대 시켰고, 중력 및 외력 시뮬레이션을 통한 Deformation 시, Fin과 Shell의 유기적인 동작을 위해 두 가지 구조를 하나로 통합하였다.

본 연구에서는 정적인 표현뿐만 아니라 중력 및 외력 시뮬레이션을 통한 움직임도 표현하므로 옷감 표현이나, 짧은 머리카락에 대한 표현, 풀밭에 대한 표현 등 다양한 분야에 응용될 수 있을 것이다.

향후 더 좋은 결과를 위해서 다음과 같은 연구들을 수행할 것이다. 사용자 측면에 있어서 첫째로, 여러 가지 장비들을 통한 사용자와의 다양한 인터랙션 방법론에 대한 연구를 수행할 것이다. 둘째로, Fur의 한 부분이 어느 방향으로 쓰러지고, 어느 방향으로 결을 이루는 등의 사용자의 요구에 적합한 Fur 데이터의 생성에 관한 연구를 수행할 것이다.

기술적인 측면에 있어서는 첫째로, Fur의 표현에 있어서 시점이 Shell에 근접해질수록 텍스쳐의 질이 낮아지기 때문에 사실감이 떨어지는 문제를 예방하기 위한 LOD 기법을 연구할 것이다. 둘째로 GPU 가속화 렌더링을 적용하여 대용량의 Mesh Data에 대한 실시간 Fur 시뮬레이션 표현을 수행할 것이다.

참고문헌

[1] J. Lengyel, E. Praun, A. Finkelstein, H. Hoppe. "Real-Time Fur over Arbitrary Surfaces" ACM Symposium on Interactive 3D Graphics 2001, pp.227-232, 2001.

[2] Dan Goldman, "Fake Fur Rendering", SIGGRAPH 97, pp.127-134, 1997.

[3] James Kajiya, Timothy Kay, "Rendering Fur with Three Dimensional Textures", ACM SIGGRAPH 89, pp.271-280, 1989.

[4] G. Yang, H. Sun, W. Wang, E. Wu, "Interactive Fur Modeling Based on Hierarchical Texture Layers", ACM VRCIA 06, pp.343-346, 2006.

[5] Microsoft Direct X 9.0 SDK, sample, "Fur".

[6] E. Praun, A. Finkelstein, H. Hoppe "Lapped Textures", ACM SIGGRAPH 00, pp.465-470, 2000.

[7] J. Dave, B. Hiebert, T.Y. Kim, I. Neulander, H, Rijpkema, W. Telford "The Chronivles of Narnia : The Lion, The Crowds and Rhythm and Hues", ACM SIGGRAPH 06, Course 11, 2006.

[8] A. Meyer, F. Neyret "Interactive Volumetric Textures", Rendering Techniques'98,Ewvgraphics Rendering Workshop, pp.157-168, 1998.