심미성 갖춘 친근한 게임 Part 2 : 효율의 극대화 Maximizing Efficiency

글 / 길라루메 프로보스트(Guillanume Provost)

이번 칼럼 시리즈의 첫 연재에서 상위 레벨에서 성능을 확인하고 레벨 설계와 환경과의 상호 작용이 어떤 영향을 미치는지 확인했다. 대부분의 이론으로 어느 정도는 설명되기 때문에 이 칼럼을 읽기 전에 지난 칼럼에 소개된 개념을 먼저 읽는 것이 좋다. '어떻게' 최적화 하는가를 결정하기 전에 언제 무엇을 최적화해야 하는지 알아야 하기 때문이다.

우리는 지난해 메쉬(mesh)가 변형의 경계나 삽입 경계로 사용될 수 있다는 것을 알았다. 본 고에서는 일반적인 하드웨어 파이프가 제공하는 보다 완전한 가능성 영역에 대해 살펴본다.

데이터에 경계가 있다면 전송된 데이터의 양도 전송 문제와(너무 많은 버텍스 등) 삽입 문제를(너무 많은 텍스처 데이터) 일으킬 수 있다. 데이터와 관련된 문제는 일반적으로 하나의 오브젝트(objects)가 고립돼 일어나는 것이 아니라 오브젝트의 집합에서 문제가 발생한다. BUS의 흐름을 방해한다는 것을 알았다면 텍스처와 버텍스를 다시 분배해야 한다. CPU에 경계가 있다면 이것은 기본적으로 처리할 수 있는 범위를 벗어난다. 프로그래밍 팀이 코드를 꼼꼼히 살펴보아야 한다.

이동 경계 메쉬의 최적화

게임에서 좀비 군대가 플레이어를 공격하도록 하려면 이들의 이동 비용을 최소화해서 이들이 경계를 넘어가지 않도록 해야 한다.

이동 경계 메쉬의 비용은 거의 버텍스 개수 × 이동의 복잡도와 같다.

따라서 우리는 이동 복잡도를 줄이거나 버텍스의 개수를 줄여야 한다. 필요하지 않은 뼈대를 줄이면 이동 복잡도를 다소 줄일 수 있지만 보다 저렴한 종류의 이동을 사용하는 것을 생각해야 한다. 몇 개의 뼈대로 타겟을 정확하게 근사화 할 수 있다면 이동 복잡도를 줄일 수 있다. 엔진이 버텍스가 하나의 뼈대에만 영향을 받는 무게가 없는 버텍스 조합으로 최적화되면 버텍스의 메쉬를 버텍스 무게가 없는 간단한 뼈대의 분배로 대체할 수도 있다. 어떤 경우라도 프로그래머와 상의해 이동 복잡도를 줄이는 데 이용할 수 있는 이동 기술이 있는지를 확인하는 시간을 갖는다.

버텍스와 메쉬

메쉬에서 버텍스를 뽑아내기 전에 비밀을 한 가지 알려 주고자 한다. 일반적인 모델링 패키지의 버텍스 개수가 현실을 반영하지는 않는다는 것이다. 이들은 파이프라인으로 이동하기 때문에 버텍스가 분리된다. 버텍스의 분리는 역으로 이동에 공간적으로 여분의 버텍스를 추가해 이동 경계 메쉬에 영향을 미친다. 이론적으로 버텍스는 삼각형을 이룰 때마다 분리될 수 있지만 실제적으로 전체 버텍스 개수는 일반적으로 두 배나 세 배가 된다. 이것을 제대로 처리하면 분리율을 대폭 낮춰 하나의 버텍스를 제거하지 않고도 메쉬의 성능을 높일 수 있다.

분리의 특성에 대해 먼저 생각해 보자. 앞에서 언급한 것과 같이

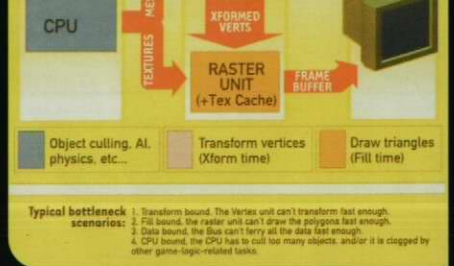
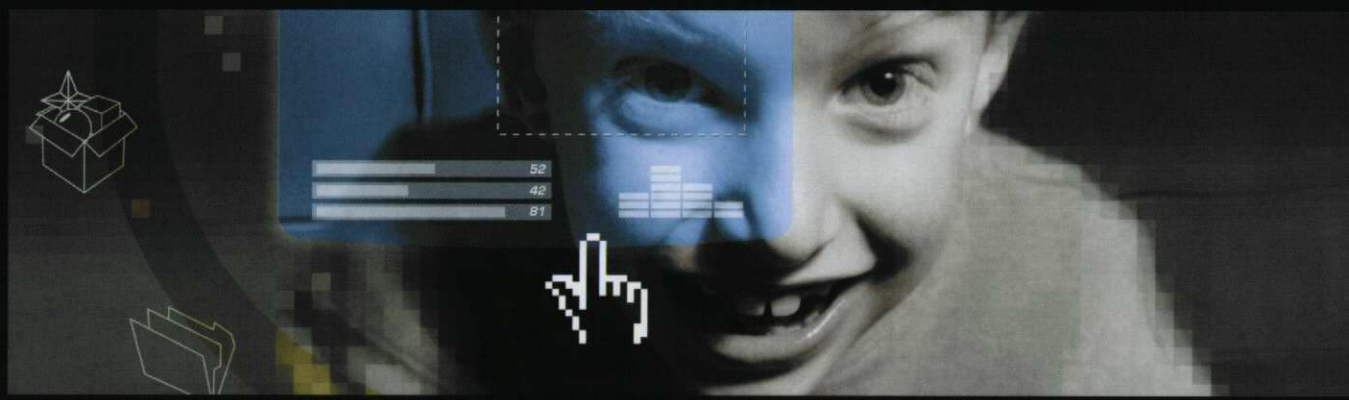


그림 1. 불규칙 하드웨어 렌더링 파이프라인 구조와 관련된 병목 현상



그래픽 하드웨어는 오브젝트가 아닌 표면1의 측면에서 생각한다. 따라서 '분리' 되는 첫 번째 버텍스는 두 개의 다른 표면에 놓인다. 머리에서 생각한 것과 같이 버텍스는 여러 자원이 공유할 수 없다.

이와 유사하게 디자이너는 일반적으로 버텍스가 다각형을 다른 평탄화 그룹이나 다른 삼각형에 다른 UV 좌표를 갖는 버텍스와 공유하도록 하지 않는다. 따라서 두 개의 다른 평탄화 그룹 경계에 놓인 버텍스와 여러 개의 UV 좌표축을 갖는 버텍스도(UV 영역이 불연속 된 경계에 놓인다) 분리2를 일으킬 수 있다.

각각의 분리를 간단히 최소화 할 수 있는 몇 가지 방법이 있다. 예를 들어, 텍스처를 지능적으로 결합해 연결하면 재료에 따른 분리를 최소화 할 수 있다.

UV 영역이 불연속이면 문제가 다소 복잡해진다. UV가 끊어지지 않도록 요소를 맵핑하려면 대칭축이나 최소 메쉬에서 '부하 지점'을 찾아야 한다.

평면이나 원통형, 입방체 맵핑 같은 맵핑 발생기를 사용하면 UV 영역 불연속점을 최소화하거나 제거할 수 있다. 예를 들어 볼로 연결된 엉덩이나 어깨, 팔이나 다리가 이러한 기술에 이용할 수 있다.

3D Studio Max 5.0과 Maya의 UV 영역에서 메쉬를 분리하려

그림 2. 버텍스 분리점이 UV 불연속점을 모아 그룹의 경계와 재료의 경계를 부드럽게 한다.

면 손상3을 최소할 수 있도록 UV를 연결할 수 있는 UV 맵핑 도구를 이용한다. 이것은 텍스처의 패스를 간략화 할 뿐만 아니라 오브젝트에 실제로 필요한 텍스처 영역을 최소화 할 수 있기 때문에 UV 영역에서 맵핑을 최적화하는데 시간을 투자할만한 가치가 있다. 대칭축이 존재하지 않는다면 사람을 처리할 때 UV 분리를 최소화 할 수 있도록 '재단' 할 천 조각에서 처리할 텍스처를 찾을 수 있을 것이다.

성능이 중요한 메쉬를 만든다면 손으로 평탄화 그룹을 미세 조정해 최적화하는 것이 가장 좋을 것이다. 목표는 서로 다른 평탄화 그룹 개수를 최소화하는 것이 아니라 이러한 평탄화 그룹을 분리하는 경계의 개수를 최소화하는 것임을 기억해야 한다. 또한 필요한 수준의 화질을 얻지 못한다고 하더라도 분리되지 않도록 적용된 텍스처를 사용해 평탄화 그룹을 '위장' 할 수 있다.

함께 묶는 다른 방법은 버텍스 분리를 '재사용' 하는 것이다. 예를 들면 앞에서 디자이너가 버텍스당 하나의 자원을 사용하고 버텍스당 하나의 평탄화 그룹을 사용하도록 한 것이 그 방법이다. 다시 말해, 같은 세트의 얼굴을 표현하는 평탄화 그룹과 자원 아이디 그룹을 가지고 있다면 이들은 한번만 분리될 수 있다. UV 불연속의 경우도 마찬가지이다. 평탄화 그룹의 경계에서 발생하면 별도의 분리가 발생하지는 않는다.

기록을 위해 메쉬가 결정적으로 이동 경계에 있다면 텍스터 메모리를 절약하는 것 보다는 버텍스 분리를 줄이는 것이 보다 중요하다. 자원이나 UV 불연속을 이용하는 각각의 발산 컬러를 제거하기 위해 메쉬를 위한 여분의 텍스처가 필요하다면 트레이드 오프를 생각해야 한다.

이것은 게임 컨텐트를 표시하기 위해 아주 상세한 메쉬를 사용하는 개념을 맵핑시키고 일반화한다. 일반적인 맵은 각각의 텍스처가 컬러가 아닌 일반적인 값을 나타낸다. 이것은 메쉬를 매우 미세하게 조정하기 때문에 평탄화 그룹을 복사해 이를 사용하기 위해 필요한 많은 세부 정보에 첨가할 수 있다. 일반적인 맵은 보통 기존의 확산 텍스처와 같은 UV 축 셋을 사용해 맵핑되기 때문에 다른 버텍스가 분리되도록 하지 않으며 이동 경계 메쉬를 평탄화 그룹보다 훨씬 쉽

그림 3. 텍스처는 천 조각처럼 텍스처를 감싸는 "바느질 솔기" 라인을 따라 메쉬에 싸이지 않는다. 이런 방법으로 메쉬에 시각적으로 제한을 가하지 않고 UV 불연속점을(붉은 색으로 표시) 최소화한다.

고 무리 없이 수행할 수 있다.

하지만 불행히도, 일반적인 맵은 손으로 그릴 수 없다. 맵의 모든 기능을 완전히 이용하고 싶다면 보다 고해상도의 자세한 메쉬를 발생시키기 위해 특별한 툴이 필요하다. 이를 지원하기 위해 필요한 픽셀 동작 때문에 모든 하드웨어 플랫폼을 지원할 수는 없다.

마지막에는 각각의 얼굴이 서로 다른 평탄화 그룹을 갖도록 해 전체 오브젝트를 평탄하게 만들어 프로그래머가 특별히 계속 지원하지 않으면 자원 사이에 일정한 사이클을 유지해야 하는 확인 보드 같은 자원 스위치를 절대 피해야 한다. 제대로 벗기면 GPU가 이를 처리할 수 있다.

본고를 쓰기 시작했을 때는 중요한 시스템 사양에 대한 가이드라인 없이 모든 메인스트림 콘솔 시스템과 최근의 모든 PC 기반 비디오 카드를 커버할 수 있을 것이라고 생각할 만큼 뭘 몰랐다. 하지만 현실적으로 이것은 너무나도 낙관적인 생각이었다.

몇몇 시스템은 색인을 갖는 요소들을 지원하지 않으며 몇몇은 T&L 이동 캐시도 없다. 어떤 경우에도 표면 이동 비용은 '제거 친화도'에 많이 영향을 받는다. 하드웨어가 이 둘을 모두 지원한다면 제거 친화도는 성능에 별 영향을 미치지 않는다.

삼각 스트립은 하나 이상의 삼각형을 공유하고 있을 때 버텍스를 여러 번 이동하지 않도록 하기 위해서 몇몇 시스템을 삼각형으로 나타낸다. 삼각 스트립에서 이후에 이어지는 버텍스가 아닌 삼각형을 구성하는 첫 번째 버텍스는 앞의 두 삼각형을 형성한다. 그래픽 프로세서가 이 스트립을 그릴 때는 마지막(그리고 다음) 삼각형의 버텍스 이동 비용을 효율적으로 공유하기 위해 삼각형 당 추가 버텍스를 이동할 때만 필요하다.

새로운 삼각형을 만들 때 선택할 수 있는 버텍스가 없는 경우 스트립 알고리즘은 '스트립을 닫는다(이동 시간을 효율적으로 늘린다).' 이 같은 상황은 보통 하나의 버텍스가 많은 개수(8개 이상)의 삼각형을 공유하고 있는 부하 지점에서 발생한다.

부하 지점은 보통 매우 얇은 삼각형 시리즈에 연결되기 때문에 얇은 삼각형을 피하고 메쉬 표면에서 버텍스 밀도를 가능한 일정하게 유지하면 일반적으로 스트립 처리 과정에 도움이 된다.

좋은 삼각 스트립 알고리즘은 대부분 같은 평면5에 놓이도록 삼각형을 자동으로 다시 배치하지만 다른 평면에 얼굴을 엮는 모서리를 다시 적응시키지는 못한다.

이동 경계 메쉬

자신이 하고 있는 작업을 지능적으로 처리한다면 이러한 기술적인 세부 사항이 이동 경계 메쉬를 세 배 이상 효율적으로 만들 수 있지만 이럴 경우 작업량이 너무 많아진다. 어려운 작업을 수행하기 전에 메쉬를 최적화 할 필요가 있는지를 항상 생각해 보아야 한다. 그렇지 않으면 기회가 생길 때만 이용할 수 있다. 마지막으로 버텍스 분리가 발생한 지점을 볼 수 있게 하는 도구를 사용하면 진정으로 최적화된 메쉬와 같은 역할을 한다. 요약하면 이동 경계 메쉬의 최적화 확인 목록은 다음과 같다.

- 오브젝트에 하나 이상의 LOD(Level of Detail) 메쉬를 만든다.
- 가능한 한 적은 뼈대와 버텍스를 사용해 이동 복잡도를 줄인다.
- 가능한 한 적은 자원 표면을 사용한다.
- 몇 개의 서로 다른 확산 컬러 대신 메쉬의 텍스처를 이용한다.
- UV 불연속을 최소화 할 수 있는 UV 발생기를 사용한다.
- 실제로 필요하지 않은 평탄화 그룹 불연속 지점을 제거하거나 이를 대신할 수 있는 이산(discrete) 컬러 변화를 이용하거나 일반적인 맵을 사용한다.
- 남아 있는 자원의 경계와 UV 영역 경계, 평탄화 그룹 경계를 맞춘다.
- 눈에 보이지 않는 모서리와 부하 지점을 확인한다.

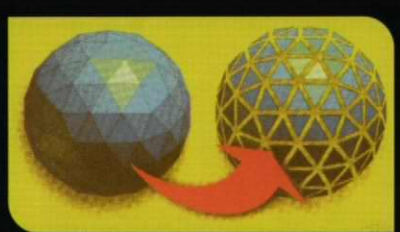


그림 4. 평면 음영이 메쉬 내부의 모든 얼굴을 분리된 평탄화 그룹에 묶어 최악의 분리 시나리오가 발생할 수 있다. 특별히 이것이 지원되지 않는다면 가능한 피하도록 한다.

• 삼각형을 비파고 메쉬 표면에서 버텍스의 밀도를 가능한 한 일정하게 유지한다.

메쉬가 이동 경계가 아니라 삽입 경계라면 앞에서 언급한 것을 수행하지 않는다. 예를 들면 삽입 경계 메쉬에 적용된 하나의 텍스처에 자원을 결합하면 실제로 캐시 미스가 너무 자주 발생해 성능이 떨어질 수 있다.

삽입 경계 메쉬의 최적화

앞에서 삽입 경계 메쉬를 그리는 것과 관련된 비용은 다음과 같은 세 가지 요소의 함수라는 것을 알았다.

삽입 비용은 거의 픽셀 크기×그림 복잡도×텍스처의 밀도와 같다

정확히 크기를 작게 할 수는 없지만 같은 가시 영역 내에 몇 개의 큰 표면을 겹치지 않도록 해야 한다. 이것의 전형적인 예는 수족관으로 전체 방의 벽을 채우거나(뒷벽과 유리 창문은 두 계층을 형성한다) 구름 낀 날씨를 표현하기 위해 하늘과 같은 계층을 연속적으로 구성한 경우이다. 투명하고 첨가된 평면은 화면에 축적돼 디자이너가 그려야 할 평면 계층을 만들거나 삽입과 관련 병목지점이 될 수 있다.

파이프라인이 양면 자원을 지원한다면 넓은 표면에 사용할 때 조심해야 한다. 디자이너가 제거돼야 하는 조각을 그리도록 한다면 디자인 비용이 크게 늘어난다. 어떤 플랫폼에서 뒷면에서 골라내는 것은 그리기 과정의 통합 과정이 아니며, 각각의 다각형을 골라내는 것은 매우 까다로운 작업이다. 이러한 플랫폼에서 저작권을 가진 콘텐츠를 사용한다면 뒷면이 필요 없는 벽이 전혀 발생하지 않도록 해야 한다.

삼각형이 커지면 커질수록 해결해야 할 텍스처 영역은 많아진다. 불행히도, 실제 화면 영역의 대부분을 차지하는 메쉬는 대부분의 텍스처 영역도 차지하며 삽입 관련 병목을 일으키는 주범이다.

텍스처 영역을 최소화하기 위해서는 두 가지를 수행해야 한다. 밉-맵을 사용하거나 발생시켜야 하며, 텍스처 형식과 크기를 지능적으로 선택해야 한다.

간단한 표를 보면 텍스처 형식을 보다 지능적으로만 선택해도 상당히 노력을 절감할 수 있음을 알 수 있다. 텍스처가 32 × 32보다 작으면 업로드와 팔레트 구성과 관련된 비용이 팔레트를 이용하지 않는 버전보다 많이 들기 때문에 팔레트를 이용하는 것은 그리 좋은 방법이 아니다. 하드웨어가 DXT1(DirectX Texture Compression) 같은 기본 압축을 지원한다면 팔레트를 이용하는 것도 좋은 방법일 수 있다.

삽입 경계 표면에 확산 컬러만 사용하지 않는다면 더 좋다. 몇몇 플랫폼에서는 텍스처가 아닌 표면에 그리는 것이 텍스처에 그리는 것보다 빠르다.

앞에서 언급했던 것과 같이 일반적인 법칙은 이동 경계 메쉬에서

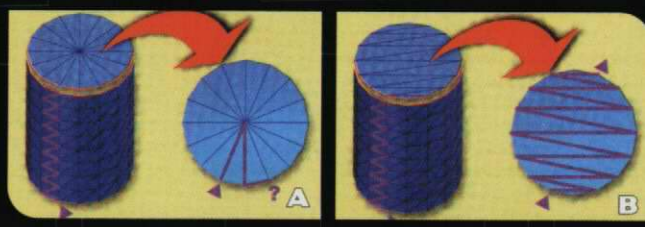


그림 5. 상위의 실린더 캡이 부적절하게 구성돼 스트립이 너무 일찍 "중지" 된다. 스트립은 평탄화 그룹 분리 때문에 실린더의 몸체를 가로지를 수 없다. 아래쪽의 캡은 다시 적절히 구성돼 단일 스트립에 완벽히 부합한다.

UV가 분리되지 않도록 텍스처 영역을 포기하는 것과 트레이드 오프를 따져보는 것이다. 메쉬가 삽입 경계라면 반대의 법칙이 적용된다. UV 영역에서 버텍스를 분리하면 텍스처 영역을 줄여 좋은 트레이드 오프 관계가 된다.

마지막으로, 삽입 경계 메쉬에 적용한 자원을 신중하게 선택하면 성능을 향상시킬 수 있다. 텍스처 패스의 개수와 자원 특정의 복잡도는 항상 삽입 경계 표면을 처리할 때 가장 중요한 요소이다.

보트를 잃은 텍스처

우리 중 몇몇은 하드웨어를 처리할 때 '크림' 문제를 해결해야 하지만 우리 대다수에게 필요한 것은 현실적인 시장에 만족하는 것이다. 콘솔 시장에서, 팀은 시스템을 한계까지 이끌어야 하지만 매우 중요한 때에 이러한 한계에 갇혀버릴 수도 있다.

이러한 상황에 처했다면 텍스처 캐시의 일관성을 생각해야 한다.

그래픽 프로세서는 일반적으로 화면 영역에 구성하는 선형, 수평 픽셀 스트립을 삽입해 삼각형을 그린다. 현재 거의 모든 하드웨어는 한 번에 여러 픽셀을 '찍어' 구현해 삼각형을 삽입하는 데 필요한 시간을 줄인다.

카드가 그리는 모든 텍스처 픽셀에는 관련된 텍스처에서 특정한 양의 텍스처를 절약해야 한다. 이것은 카드가 텍스처 블록을 붙일 수 있는 기본적인 스크래치패드인 텍스처 캐시를 통해 수행된다. 카드가 새로운 픽셀을 그릴 때마다 캐시를 확인해 캐시가 필요한 텍스처가 이미 현재 '스크래치 패드' 에 존재한다면 모든 작업은 정지 없

Size/ Format	16-color PAL	256-color PAL	16-bit RGB	32-color ARGB	DXT1 RGB
32X32	Do not use	Do not use	2K	4K	1K
64X64	2K	4K	8K	16K	4K
128X128	8K	16K	32K	64K	16K
256X256	32K	64K	128K	256K	64K

표1. 텍스처 형식을 제대로 선택함으로써 줄일 수 있는 작업

이 실행된다. 캐시를 필요로 하는 텍스처가 캐시 안에 없다면 카드는 새로운 텍스처 뭉치를 읽어 그리기 작업 전에 캐시에 저장한다. 이것이 텍스처 캐시 미스이다.

텍스처 캐시 일관성이 좋다는 것은 표면을 그릴 때 텍스터 캐시 미스가 거의 발생하지 않는다는 것을 의미한다. 텍스처 캐시 일관성이 좋지 않으면 표면을 그리는 데 시간이 많이 필요하다. 대부분의 PC 기반 시스템과 현재의 몇몇 고성능 콘솔은 그리는 모든 픽셀에서 적절한 밉을 선택해 자동으로 텍스처 캐시 일관성을 유지한다. 하지만 다른 시스템은 도형 메쉬의 표면 영역이 텍스처 밀도가 픽셀을 제대로 선택한다면 일정하다는 사실에만 의존한다.

이러한 시스템에서 텍스처의 밀도가 균일하지 않으면 픽셀에서 픽셀로, 텍스처 영역에서 카드로 '점프'하게 된다. 그렇게 되면 심각한 텍스처 에일리어싱(aliasing) 문제가 발생하고 계속해서 텍스처 캐시 미스가 발생해 카드가 '스크래치패드'에 없는 텍스처를 패치한다.

디자이너는 메쉬에 텍스처의 밀도를 균일하게 해 시각적 예술성(artifacts)과 성능 문제를 해결할 수 있다. UV 영역에서 얼굴 크기와 모양이 거의 도형의 해당 부분에 비례하도록 해 이를 처리한다. 이것은 디자인 측면에서도 가능한 개념이다. 얼굴이 커지면 작은 얼굴보다 세부 정보를(보다 큰 UV 영역) 저장하기 위해 보다 많은 텍스처가 필요하다.

이 개념은 오브젝트에도 적용된다. 오브젝트가 작으면 화면에 표시되는 크기도 작고(세부 정보가 적기 때문에) 텍스처의 크기도 작다.

삽입 경계 표면

다음은 삽입 경계 도형을 만들 때 고려해야 할 사항이다.

- 모든 텍스처를 위한 밉-맵 구성
- 가능하면 복잡한 자원 특성을 (범프 맵, 광택이 나는 재료 등) 요구하는 큰 표면을 사용하지 않는다.
- 큰 투명 계층이나 첨가 계층을 많이 겹쳐서 사용하지 않는다.
- 반드시 해야 하는 경우가 아니라면 양면의 큰 벽/천장 조각을 사용하지 않는다. 엔진이 뒷면 제거 기능을 지원하지 않는다면 불필요한 큰 뒷면을 반드시 제거한다.
- 텍스처 영역을 절약하려면 텍스처 형식을 신중하게 선택한다. DXT1 같은 압축 형식을 액세스 할 필요가 없다면 팔렛트 텍스처를 이용해야 하는지 확인한다.
- 버텍스를 분리해야 하더라도 큰 텍스처보다는 작은 조각이나 확산 물질을 사용한다.
- 표면에 텍스처 밀도를 가능한 균일하게 분배할 수 있도록 UV 맵을 조정한다.

삽입 경계 표면에서 다행한 일은 많은 버텍스를 추가하는 것이 도

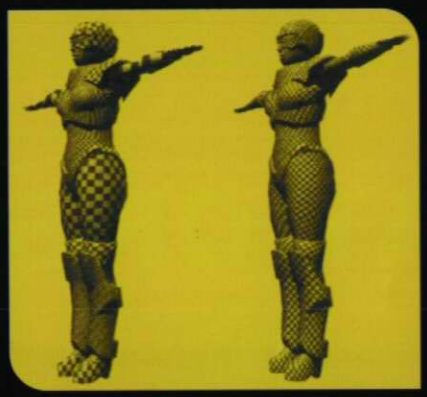


그림 6. 왼쪽의 비균일한 텍스처 밀도는 특정 플랫폼에 시각적 인공물을 만든다. 오른쪽의 텍스처 밀도는 기하학적 공간 기능으로 균일하다. 이것이 채움 경계 메쉬였다면 텍스처 밀도가 균일하지 않을 경우 특정 플랫폼에서 캐시 미스가 발생해 메쉬의 전체 채움 시간이 증가하게 된다.

움이 크게 되지는 않는다고 하더라도 메쉬가 이동 경계가 될 만큼 버텍스 밀도가 높아질 때까지는 큰 영향을 주지 않는다는 것이다.

효율적인 콘텐츠 작성해야

지금 머리속이 복잡하다면 '어떤 경우에도 당황하지 말자'라는 더글라스 애덤스(Douglas Adams)의 모토를 떠올려보자. 단순히 눈을 만족시키는 것보다 성능 친화 콘텐츠가 해야 할 일이 많다고 하더라도 효율적인 콘텐츠를 작성하는 것은 직관적이며 자연스런 과정이다.

버텍스나 텍스처, 오브젝트 중 어느 것도 세부적인 것을 빼내는 것보다는 균일하게 배치하는 것이 중요하다. 이것은 매우 직관적인 개념이다. 화면에 작게 표시되는 것은 세부 정보가 적고 화면에 크게 표시되는 것은 정보량이 많다.

프로그래머는 항상 조금이라도 빠르게 동작시킬 수 있도록 코드를 최적화한다. 하지만 화질을 손상보지 않고는 해결할 수 없는 하드웨어의 한계도 있다. 시스템의 한계를 극복하려면 해결책은 코드가 아닌 게임의 프레임 율을 구동하는 콘텐츠에 있다.