

Game Graphics

470420-1
Fall 2013
10/14/2013
Kyoung Shin Park
Multimedia Engineering
Dankook University

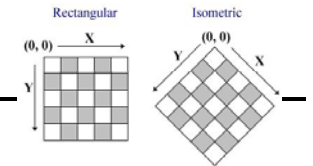
Isometric Games

Sprite-based graphics

- 스프라이트 게임 (sprite games) 은 일반적으로
 - Top down (e.g. 미로 게임)
 - Front facing (e.g. platform games)
- 작은 부분은 클리핑 (clipping) 함
- 은면제거 (hidden surface removal) 를 위해
 - 뒤에서부터 그리고 앞은 나중에 그림 (drawn from back to front)
 - 주변에 투명한 영역을 줌
- 상태정보는 오버레이 (overlay) 해서 보여주거나 다른 패널에서 보여줌
- 깊이정보는 레이어 (layers)를 사용해서 줌
 - 패럴랙스 스크롤링 (parallax scrolling) 를 사용 - 산, 울타리, 전경 등 서로 다른 깊이로 층이 된 타일을 저장하고 서로 다른 속도로 그들을 움직여서 깊이감을 전달함
 - 스프라이트 (sprites)의 크기를 조절하여 사용
- 충돌검사 (collision detection)
 - 바운딩 박스 (bounding boxes) 사용
 - 픽셀 기반 충돌검사 - 이미지 자체의 충돌을 픽셀마다 검사하는 방법. 효율이 떨어지고 느림. action이 필요할 때만 사용.

3

Basics of Isometric Games



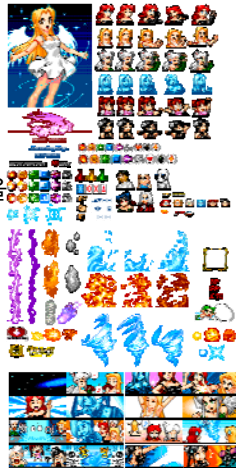
- 등축 (isometric) 게임
 - 각 물체는 지면 (x, y) 위에 위치하고 있음
 - 각 물체는 필요에 따라 지면으로부터 (z) 만큼 올라간 높이를 가짐
 - 게임세계는 등축 원근 투영으로 보여짐
 - 평행 (parallel) 투영
 - 주요 축에 대해 45 각도로 회전된 것임
 - 장점 - 모든 물체가 같은 크기로 보임. 전체적으로 보기에 좋음.
 - 단점 - 투영이 왜곡됨
 - 시점을 제한함으로써 개선시킬 수 있음
 - 수직축에 대한 왜곡 (distortion)이 심함
 - 3차원과 달리 모든 기능이 지면에서 이루어짐
 - 은면제거는 깊이 정렬 (depth sorting)을 이용함
 - 쉽게 구현이 가능함
 - 많은 게임에서 사용됨
 - Command and Conquer (except Generals) , Diablo, Age of Empires , SimCity, 등등

4

Sprites

□ 화면 위에서 유령(sprite)처럼 뒹뒹 떠다니는 것

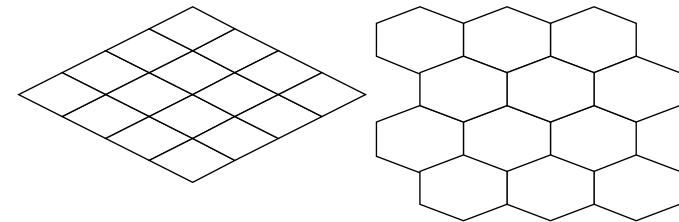
- 투명색
 - 짝하지 않는 색
 - 그림과 구별이 잘 되어야 함
 - 이미지에 사용되지 않아야 함
- 잔상이 남지 않는 그림
- 흔히들 일반적으로 게임에 쓰이는 모든 그림을 스프라이트 (sprite) 라고 부르는 경향이 있음



Tiles

□ 대부분의 등축 게임 (isometric games)은 타일 (tiles)에 기반을 두고 있음

- 원근투영으로 다이아몬드처럼 보이는 사각형
- 육각형
- (perspective가 없어서) 모든 깊이에 다 같은 고정된 크기 (fixed size)를 가짐



Tiles

□ 타일의 표면은 겉으로 타일의 구조가 보이지 않게 잘 배치 해야 함

- 주의 깊게 디자인된 타일
- 약간의 변형 (비슷한 terrain에서 더욱 중요함)
- 높이의 차이만이 보여짐 (실제로 게임에서 구현되진 않음)
- 투명하지 않으며 움직이지 않는 정적인 것임



Tiles

- 3차원의 느낌이 나도록 만듦 (주요 축이 확연히 들어나도록)
 - 그래픽에서 길이나 벽 등등
 - 움직임에서 키맵핑 (key mapping) 시
- 미리 구성될 수도 있으며, 날아다니게 만들 수도 있음
 - 배치 규칙 (alignment rules)이 필요함
 - 일반적으로 고수준의 표현으로부터 생성됨 - 길, 강, 산, 바다, 등등



Age of Wonder (2003),
a dimetric hexagonal tile-based game

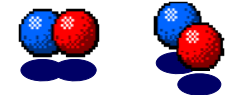
Structures on the Surface

- 지면에 고정된 것들 - 나무, 빌딩, 등
 - 2차원 지면에서 고정된 지점에 위치함
 - 종종 그리드 (grid)로 배치됨
 - 투명한 스프라이트 (sprites)
 - 애니메이션이 될 수도 있음
 - 게임에서의 물체들이거나 또는 배경 또는 전경
- 지면에서 움직이는 유닛 (moving unit) - 사람, 자동차, 등
 - 2차원 지면에서 위치로 표현됨
 - 타일 안에 고정돼 있을 필요가 없음
 - 구조나 다른 유닛 뒤로 움직일 수 있음
- 지면 위에서 움직이는 유닛 - 총알, 풍선, 등
 - 지면 위에 한 점과 높이로 표현됨
- 움직임 (movement)
 - 사용자 인터페이스 (예, 화살방향키)
 - 원하는 대각선 움직임
 - 수직, 수평의 움직임 속도를 다르게 함

9

Collision Detection

- 복잡한 3차원 계산을 피하도록 함
- 스프라이트는 중첩이 가능함 (물체는 중첩이 안됨)
- 물체의 경우, 지면에 이 물체의 그림자를 저장함
 - 혹은 약간의 대략적인 형체나 그림자의 관련된 부분을 저장함
 - 필요하면, 이 물체의 높이 간격도 저장함
- 지면 투영은 convex라 가정함 (아니면, 지면을 여러 개로 분리함)
- 충돌검사
 - 그림자와 충돌하는지 검사
 - 만약 그렇다면, 그것의 높이 간격도 충돌하는 지 검사
 - 만약 그렇다면, 충돌된 것임
 - 그러나 가끔 보다 복잡한 충돌검사가 요구됨



10

Collision Detection

- 만약 다른 장소에서 다른 높이 간격이 존재한다면, 분리함
 - 예를 들어, 왼쪽, 가운데, 오른쪽 부분으로 구성된 대문이 있다면
 - 가운데 부분에서는 충돌검사를 위한 마스크를 저장하지 않기도 함 (사람들이 단지 걸어간다고만 하면 충돌이 생기지 않을 것이기 때문에)
- 게임제작 시
 - 충돌검사를 위하여 모든 물체의 스프라이트 그림자에 대해 마스크를 지정함
 - 높이 간격은 필요에 따라 지정해야 함

11

Hidden Surface Removal

- 은면제거를 위한 그리기 순서 (drawing order)
 - 배경 (backgrounds)
 - 물체 (objects)
 - 전경 (foregrounds)
 - 중첩 (overlays)
- 각 물체는 지면에 위치를 가지고 있음
 - 위치를 잘 지정해야 함 - 예를 들어 그림자의 가운데 안에 물체를 위치시킴과 같이
- 등축관측 시 그리기 순서 (drawing order)
 - 위에서부터 아래로 위치를 정렬시킴
 - 그리고, 뒤에 멀리 있는 것부터 먼저 그리기 시작



12

Hidden Surface Removal

- 그러나, 그리기 순서가 항상 정확하지는 않음 – 특히 물체가 서로 가까이 근접했을 시 판단하기 어려움
- 그림자가 중첩됐고 높이 정보가 있을 때는
 - 높이가 증가하는 순서로 정렬함
 - 그리고, 다른 물체보다 아래쪽에 있는 것부터 먼저 그리기 시작
- 게임제작 시
 - 지면의 위치에서 스프라이트의 원점을 선택함 (스프라이트의 바깥쪽도 가능함!)
 - 깊이 정렬을 확실하게 하기 위해 각 단계마다 깊이를 -y로 지정함

13

Path Finding

- 게임세상을 셀의 배열 (an array of cells)로 저장함
- 물체 (또는 물체의 일부)를 포함하고 있다면 금지된 셀로 표시함
- 게임에서 자주 쓰이는 길찾기 (path finding problem)에 가장 범용적이고 확실한 알고리즘인 A* algorithm을 사용함



3D Graphic Games

3D Graphics

- Frame buffer and Double buffering
- Visibility and Depth buffer
- Alpha blending, Stencil buffer
- Coordinate Systems
- Triangle, Vertex, Index, Polygon, Mesh
- Transformations
- Camera
- Lightings, Shading, Materials
- Textures
- Billboarding
- Normal mapping, Bump mapping, Displacement mapping

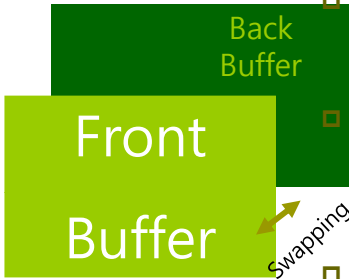
16

Frame Buffer

- 넓은 의미로 프레임 버퍼는 화면에 도시할 래스터 이미지 (raster image)일 뿐만 아니라 그러한 이미지를 생성하는데 필요한 여러 부류의 정보를 저장해주는 포괄적 의미의 그래픽스 전용 메모리를 뜻함
- 색 버퍼 (color buffer)
 - 더블 버퍼 (double buffer) – front buffer & back buffer
 - 스테레오 버퍼 (stereo buffer) – 양안 시차 이미지
 - 알파 버퍼 (alpha buffer) – 투명도
- 깊이 버퍼 (depth buffer) – 은면 제거
- 스텐실 버퍼 (stencil buffer) – 2차원 이미지와 블렌딩
- 축적 버퍼 (accumulation buffer) – 렌더링 속도 향상
 - 보통의 이미지들을 혼합된 이미지들로 **축적**하는데 사용. 사용자는 장면 안티앨리어싱과 같은 작업들을 수행할 수 있음.
- 픽셀 버퍼 (pixel buffer)

17

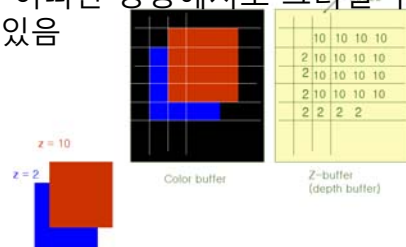
Double Buffering

- 
- 이미지가 후면버퍼 (back buffer)에 그려질 동안 전면버퍼 (front buffer)는 완성된 이미지를 viewer에게 보여줌.
 - 다음 보여질 화면을 후면 버퍼에 미리 그려놓은 다음에, 그것을 현재 화면에 보여주고 있는 전면 버퍼와 바꿔치기 (swapping buffer) 하는 방법
 - Swapping은 보통 모니터(50Hz)의 수직 공백 기간 동안 이루어짐
 - 대부분의 PC는 그래픽 감박임 결과에서 이 동기화를 사용하도록 설정하는 경향이 있음
 - Double buffering은 부드러운 애니메이션에 유용함

18

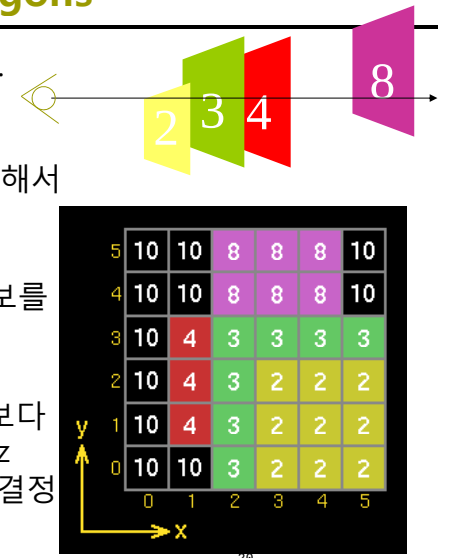
Visibility and Depth Buffer

- 색 버퍼 (color buffer)와 동일한 해상도임
- 깊이 버퍼 (z-buffer, depth buffer)는 그리고자 하는 픽셀당 깊이 정보 (depth value)를 가짐
- Z-buffer algorithm은 새로운 픽셀을 그릴 때마다, 새로운 깊이 정보를 깊이 버퍼 (z-buffer) 안에 있는 깊이(depth) 정보와 비교하여, 더 먼 것이 깊이 버퍼에 저장됨
- 다각형 (polygons)은 어떠한 방향에서도 그려질 수 있으며 교차할 수도 있음



Depth Sorting of Polygons

- PS1는 Z-Buffering 하지 않음. 따라서, polygon은 전향순서 (back-to-front order)로 정렬 (depth sorting of polygons) 해서 그려야 함
- Z Buffer는 보통 ~ 16-32bits/pixel. 픽셀당 깊이 정보를 저장하는 버퍼
- Z-buffer algorithm은 픽셀 단위로 어느 픽셀이 다른 것보다 앞에 있는 지 판단하여 작은 z 값을 가진 것이 앞에 있는지 결정
- 래스터 시 polygon의 깊이 정렬화가 필요 없음



20

Alpha Buffer

- RGBA – alpha는 4번째 색으로 불투명도(opacity of color) 조절에 사용함
 - 불투명도 (opacity)는 얼마나 많은 빛이 면을 관통하는가의 척도임
 - 투명도 (transparency)는 $1 - \text{alpha}$ 로 주어짐
 - Alpha=1.0 – 완전히 불투명
 - Alpha=0.5 – 반투명
 - Alpha=0.0 – 완전히 투명
- 알파 블렌딩 – 물체의 색을 투명하게 나타나게 함

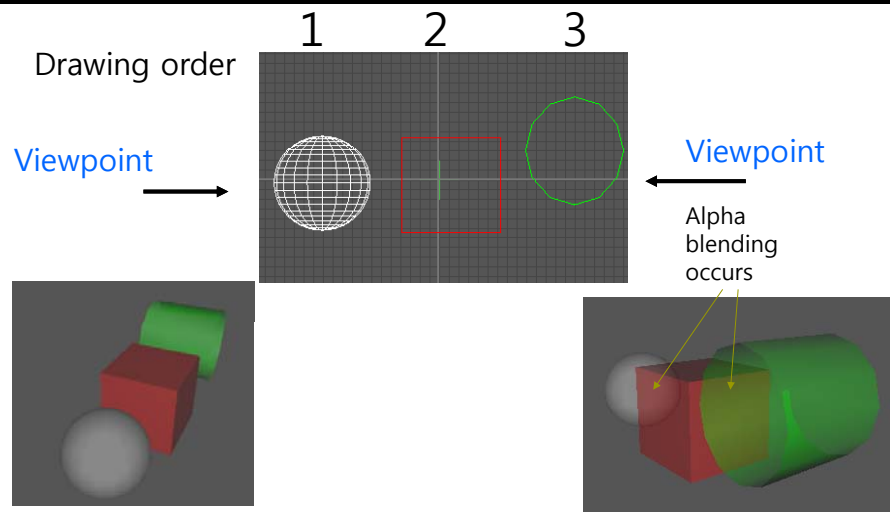
21

Blending and Drawing Order

- 블렌딩은 현재 그리고자 하는 물체와 이전에 그려진 물체의 그림 그리는 순서(drawing order)가 중요함
 - 블렌딩 함수의 source color(현재 그리고자 하는 물체의 색)와 destination color (이미 그려진 프레임버퍼의 색)로 작용함
- 만약 투명한 물체와 불투명한 물체를 같이 그리고자 한다면, 불투명부터 먼저 그린 후에 투명한 것을 그릴 것
 - Depth-buffer가 블렌딩 전에 실행되도록 함
- 만약 여러 개의 투명한 물체를 같이 그리고자 한다면, 전향 순서 (back-to-front order)로 그릴 것
 - 이 순서는 카메라의 위치에 의해서 달라질 수 있음
- 여러 개의 투명한 물체를 같이 그릴 때, 서로를 가리는 현상 (occlusion)을 막기 위해서 depth mask를 비활성화함
 - 깊이버퍼를 read-only로 만듦

22

Blending and Drawing Order

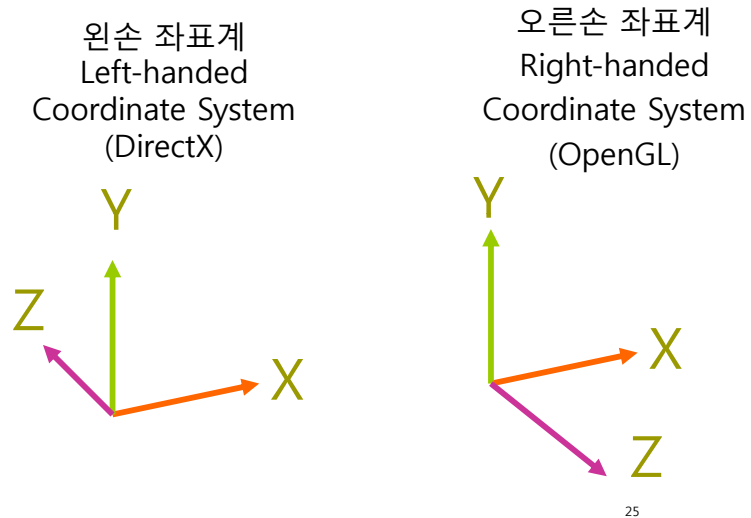


Stencil Buffer

- 특수한 효과를 위한 off-screen buffer
- Back buffer, depth buffer와 동일한 해상도임
- Stencil buffer를 사용하여 back buffer의 일정 부분이 렌더링 되지 않도록 함 – 스텐실 참조값 (reference value)와 마스크 (mask)를 비교 (comparison)하여 특정 픽셀의 렌더링 여부를 결정
- 마스크 (masking) 또는 컬링 (culling)에 주로 사용
- 대표적으로는 거울 (mirror), 그림자 (shadow) 구현에 사용
- 예를 들어, 벽면에 거울이 있는 경우, 벽면을 제외하고 거울이 있는 영역에 대해서만 반사되는 물체의 drawing을 수행하도록 함

24

Coordinate Systems



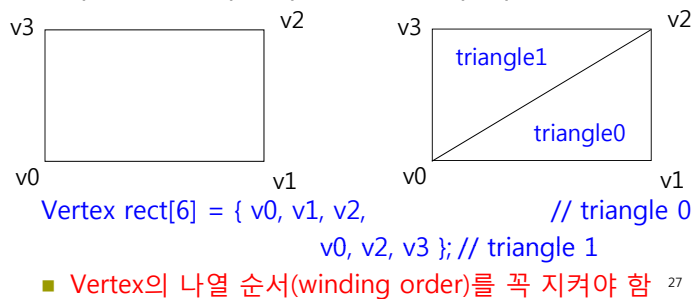
Vertex

- 정점 (vertex)은 공간적 위치
- 부가적 특성
 - 표면의 법선 벡터 (surface normal)
 - 색 (color)
 - 텍스처 좌표 (texture coordinate)
 - 그 외에 데이터 셰이더 프로그램에서 필요한 정보들..
- 한 삼각형 (triangle)은 세 개의 점 (vertices)를 사용
 - 인접한 삼각형 (adjacent triangle)에 대한 정점은 공유됨

26

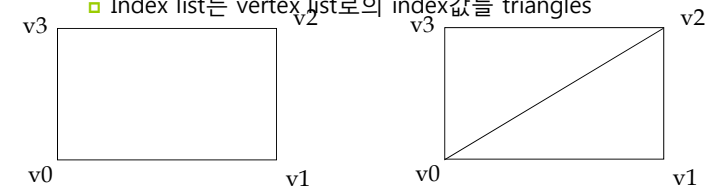
Triangle

- 3차원 물체의 기본 구성 요소 (fundamental primitive)
 - 선과 스프라이트만 제외한 폴리곤 등 모든 물체는 삼각형(Triangle)으로 만들어질 수 있음
 - 예를 들어, 사각형은 두 개의 삼각형으로 표현할 수 있음
- 세 점으로 한 평면을 형성함
- 삼각 평면은 텍스처 (textures)와 색 (colors) 정보와 연결됨



Index

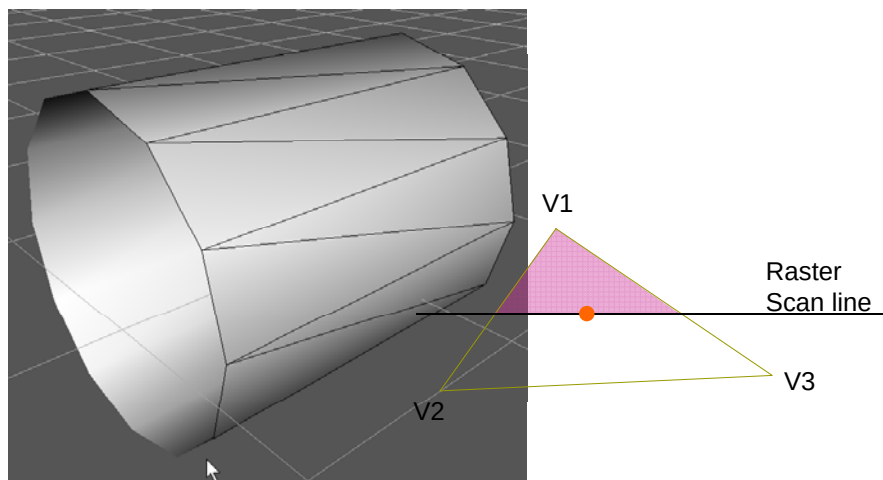
- Index
 - 만약 index list가 사용하지 않을 경우, 너무 많은 정점(vertices)이 중복됨
 - 중복된 정점을 줄이기 위해 vertex list와 index list를 사용함
 - Vertex list는 모든 vertex들
 - Index list는 vertex list로의 index값들 triangles



```
Vertex vertexList[4] = { v0, v1, v2, v3 }; // vertex list
int indexLIST[6] = { 0, 1, 2,
                   0, 2, 3 }; // index list
```

28

Polygons, Meshes

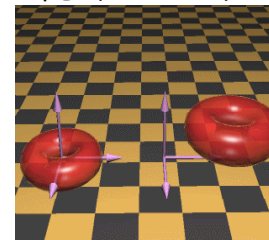


29

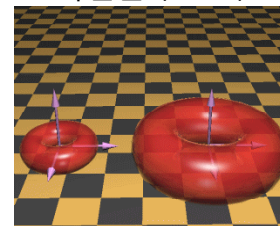
Transformations

- 변환은 좌표계의 원점을 중심으로 일어남

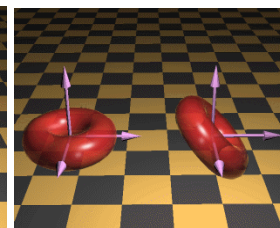
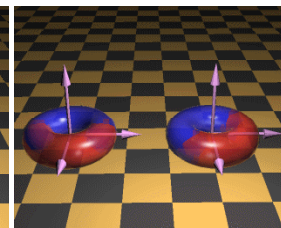
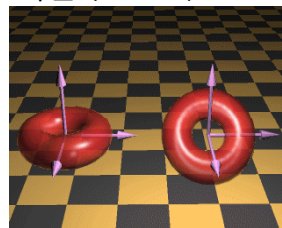
이동 (Translate)



크기 변환 (Scale)



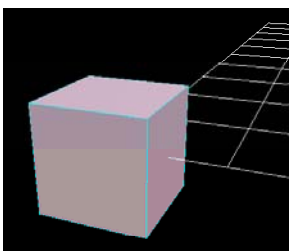
회전 (Rotate)



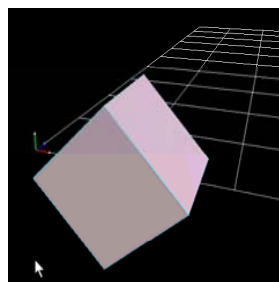
30

Transformation Order Matters

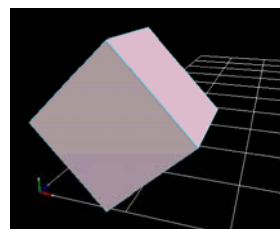
Box centered at origin



Rotate about Z 45;
Translate along X 1



Translate along X 1;
Rotate about Z 45

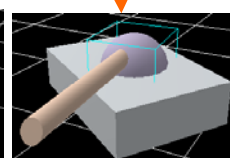
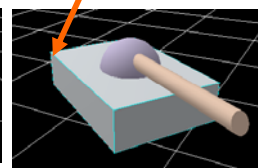
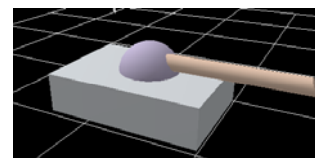
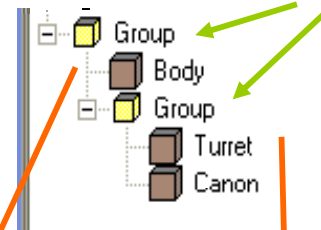


31

Hierarchy of Coordinate Systems

- 흔히 부르길,
 - Scene graph
 - Parent and child entities

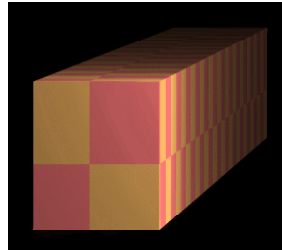
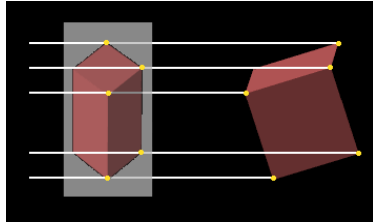
Local coordinate system



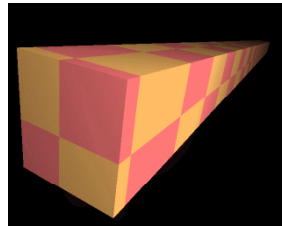
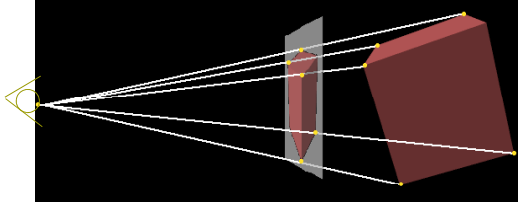
32

Camera – Parallel vs. Perspective Projection

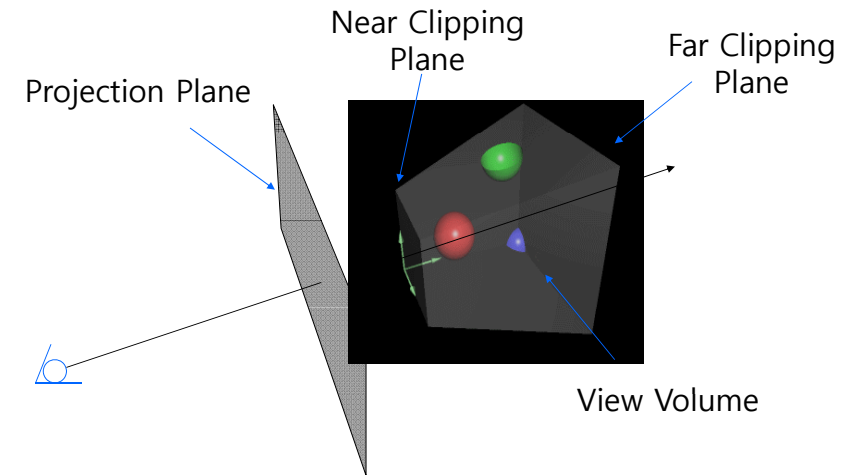
평행투영 (Parallel Projection)



원근투영 (Perspective Projection)



Camera – Perspective Projection



34

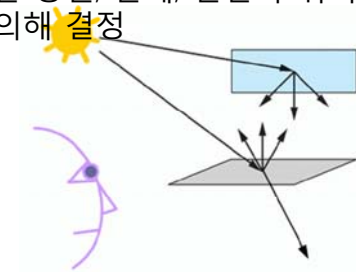
Camera Movement

- 1인칭 시점 카메라 (First person camera)
 - 게이머가 카메라를 직접 제어함 (캐릭터에 의한 것이 아님)
 - 지면으로부터 높이는 일반적으로 시스템에 의하여 이루어짐
 - Twist는 일반적으로 사용되지 않음 (Descent 게임을 제외하고)
 - View volume은 원근투영에 중요함
 - 모니터 뒤의 게이머의 위치에 의해 영향을 받음
 - zoom-in 또는 zoom-out은 부자연스러우며 위험스러움
- 3인칭 시점 카메라 (Third person camera)
 - 캐릭터로부터 일정한 거리에 위치함
 - 움직임에는 약간의 lag이 있어야 함
 - 캐릭터가 움직이지 않거나 게이머가 제어할 때만 회전이 가능하도록 함
 - 카메라의 up-vector (즉, twist) 에 조심할 것
 - 장애물이 있을 시 조심할 것

35

Lighting

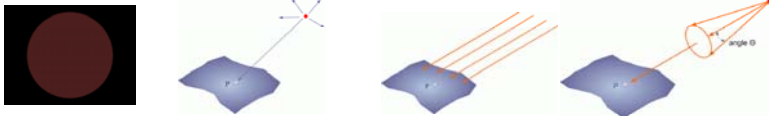
- 광원 (lighting source)에서 출발
- 물체 표면에서 재질(material)에 따라
 - 흡수 (absorption)
 - 반사 (reflection)
 - 투과 (transmission) 또는 굴절 (refraction)
- 물체를 본다는 것은 우리 눈으로 입사하는 빛에 의함
- 물체 색은 광원, 물체, 관찰자 위치, 광원과 물체의 특성에 의해 결정



36

Light Source

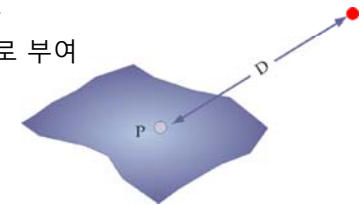
- 일반적인 광원 (light source)은 매우 작업하기 어렵다
왜냐하면 광원으로부터 나온 모든 빛에 대한 통합을
해야하기 때문에.
- CG에서는 간단한 광원 (light source)을 사용함
 - 환경광원 (Ambient light) – 장면에서 언제 어디서나 같은 양의
빛을 제공
 - 점광원 (Point light) – 위치 (position)와 색 (color)을 모델
 - 방향성광원 (Directional light) – 방향 (direction)을 모델 (parallel
light)
 - 점적광원 (Spot light) – 이상적인 점광원에서 빛의 방향을 제한



37

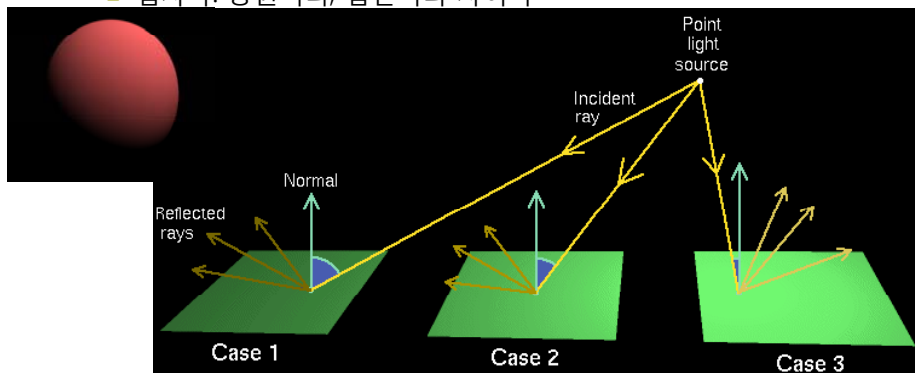
Ambient Reflection

- 광원에 직접 노출되지 않는 면에 밝기를 부여
- 모든 빛의 경로를 추적하기 어려움
 - 면마다 상수 크기의 밝기를 추가
 - 전역 조명모델 효과를 근사적으로 부여



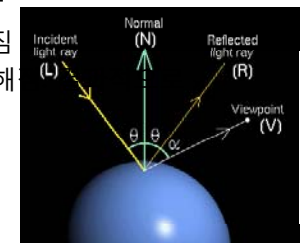
Diffuse Reflection

- 면의 방향에 따라 차등적 밝기로 입체감 부여
- 램버트 법칙(Lambertian Law)
 - 면의 밝기는 입사각의 코사인에 정비례. 입사각이 0도 일 때 가장
많이 반사되고 90도 일 때 반사되지 않음.
 - 입사각: 광원벡터, 법선벡터 사이각



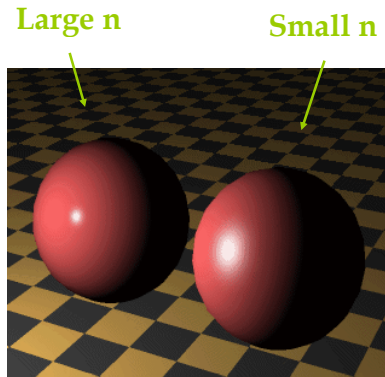
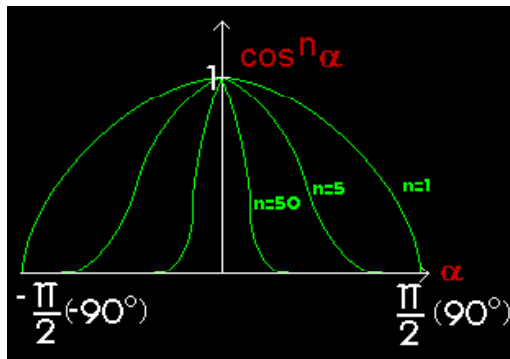
Specular Reflection (Phong Lighting Model)

- 반질반질한 표면에서 반사되는 빛
- 가장 많은 정반사는 시점 (viewpoint)이 빛의 방향에
정확히 반대 방향일 때 생김 (즉, alpha가 0도 일 때)
 - 정반사 (specular reflection)는 alpha가 커질 수록 급격하게
감쇠함
- 빛의 세기 감쇠는 $\cos^n(\alpha)$ 로 근사적으로 계산
 - 재질에 따라 n은 1에서 수백 사이로 지정함
 - $n=1$ 이면 넓고 부드럽게 빛의 세기가 약해짐
 - n이 커지면 작고 선명하게 빛의 세기가 약해
생성되는 하이라이트가 작게 나타남
 - 완벽한 정반사 모델은 n이 무한대임



40

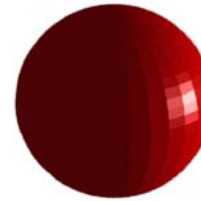
Fall off in Phong Shading



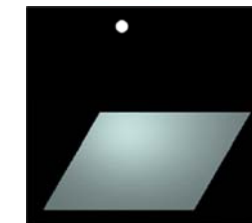
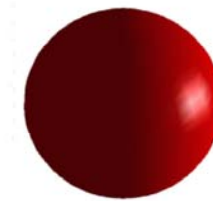
41

Flat, Gouraud, and Phong Shading

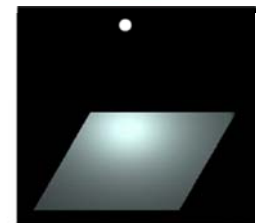
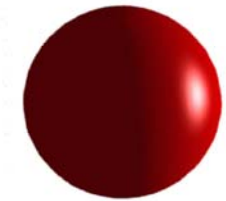
- Shading - 음영 또는 표면 렌더링(surface rendering) - 물체 면의 색을 부여



Flat shading



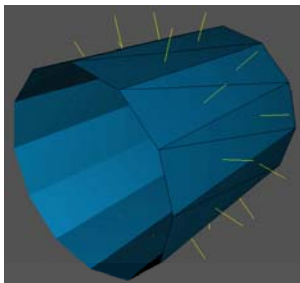
Gouraud shading



Phong shading

Flat Shading

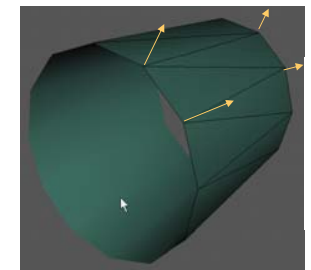
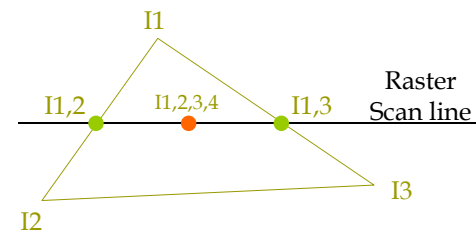
- 주어진 하나의 다각형 전체를 동일한 색으로 칠함. 빠르고 간단함.
 - 다각형을 구성하는 다각형 점들의 위치를 평균하여 중심점 (centroid)를 구하고, 중심점에서의 법선벡터, 광원벡터, 시점벡터를 기준으로 조명모델이 가해지며 그 결과 색이 면 내부를 모두 채움
 - 상수 셰이딩 (constant shading), 깎은 면 셰이딩 (facet shading)



43

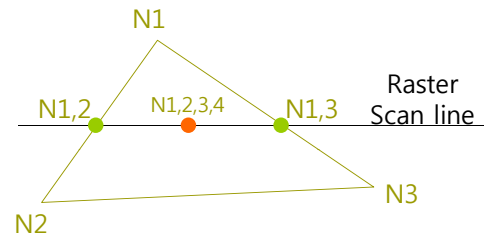
Gouraud Shading

- 정점의 색을 보간함
 - 정점의 법선 벡터를 요함. 인접면의 법선벡터를 평균하여 구함
 - 정점의 색으로부터 내부면의 색을 선형보간
- 다각형 내부를 서로 다른 색으로 채우는 방법
- 경면광 (specular highlight)을 감안하지 않음
 - 실제적인 정점의 법선벡터와 근사적으로 계산된 법선벡터가 완전히 일치하지 않기 때문



Phong Shading

- 정점의 색 대신 정점의 법선 벡터를 보간.



- 곡면의 기울기가 복원됨. 경면광 (specular highlight)을 부여할 수 있음.



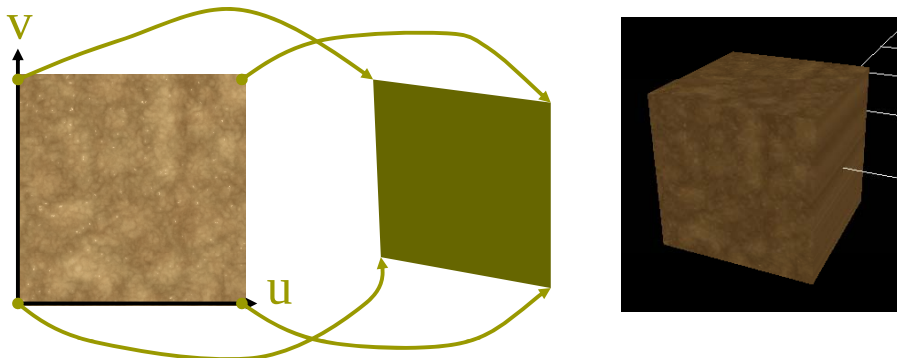
45

Texture Maps Used in Tank Game



Texture Mapping

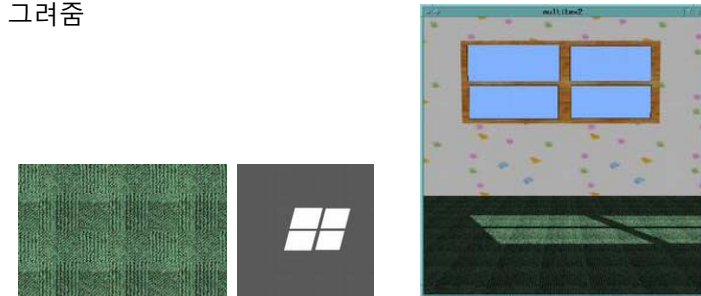
- 제한된 수의 다각형을 사용해야 하는 실시간 렌더링에 있어서 비교적 적은 추가 비용으로 이미지의 사실성을 상당히 높일 수 있는 기법임



47

Multipass Rendering

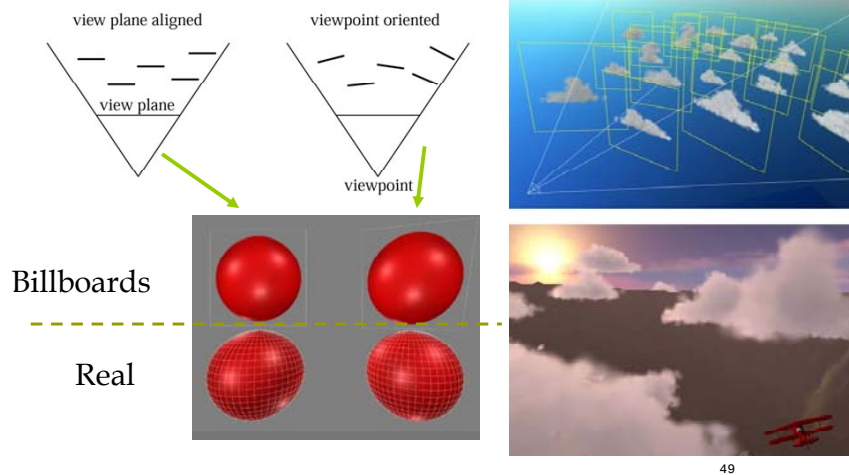
- 같은 물체를 다른 모드를 사용하여 여러 번 렌더링하는 것
 - 예를 들어, **라이트 맵 (lightmap)** 효과를 위해 물체를 정상적으로 그리고 난 후 블렌딩 함수를 사용하여 같은 물체를 다시 한번 그려줌



48

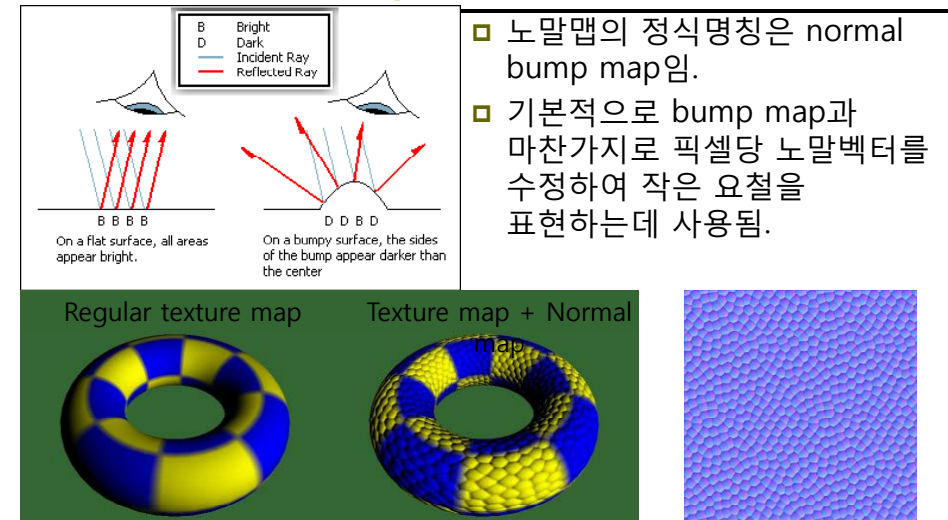
Billboarding

- 물체의 방향이 언제나 카메라를 향하도록 하는 것



49

Normal Mapping



50

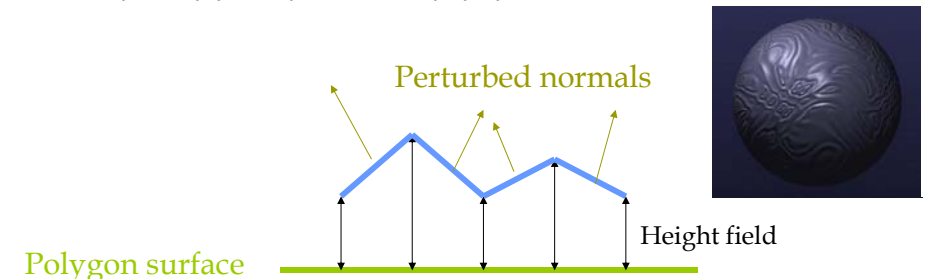
Normal Mapping

- 노말맵은 R,G,B 채널로 되어 있으며, 이 세 값은 폴리곤 표면의 tangent에 관련된 법선 벡터의 방향을 나타냄. 실제로 쓰이는 의미는 "수직" "수평" "깊이" 즉 x, y, z 축으로의 값을 표현하는 것임
- 폴리곤의 표면 상의 한 점은 최종적으로 x, y, z 축의 기울기를 모두 갖게 되는 것임
- 최근 게임에서는 노말맵이 high-polygon 장면에서 계산된 다음 low-polygon 모델에 맵핑되어 생생하게 세부묘사를 하고 있음

51

Bump Mapping

- 노말맵이 있기 전까지는 흑백의 이미지 (greyscale texture)를 사용해서 어둡고 밝은 부분에 따라 높낮이 (height field)를 조정하는 범프맵 (bump map)을 사용
- 픽셀단위로 폴리곤의 표면에서 밝은 부분은 튀어나오게 하고 어두운 부분은 들어가게 함

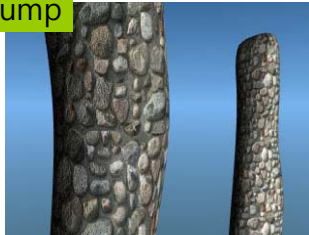


52

Displacement Mapping

- 노말맵/범프맵은 폴리곤의 옆면에서 보게 되면 편평하게 보이는 단점이 존재함
- 디스플레이스먼트 맵핑 (displacement mapping)을 사용하면 메쉬 (mesh)의 실제 geometry를 움직여서 보다 울퉁불퉁하게 보이게 할 수 있음
- 디스플레이스먼트 맵의 높낮이 (height field)는 메쉬의 geometry를 법선 벡터에 따라 옮겨놓는 데 사용

Bump



Displacement



53

Texture mapping

- 각종 텍스처 맵핑 기법
 - **Color** (diffuse reflection coefficients) - **Texture mapping**
 - **Specular color** - 반사빛 (specular light)과 반사계수 (reflection coefficients)를 시뮬레이션하는 방법
 - **Environment mapping**은 높은 반사표면 (highly specular surface)을 렌더링할 수 있는 저렴한 방법
 - **Normal vector** - 재질 (material)의 거칠기 (roughness)와 구조 (structure)를 시뮬레이션하는 방법
 - **Bump mapping/Normal mapping**은 노말벡터를 교란하여 실제같이 울퉁불퉁한 표면을 만들어냄. 그러나 표면의 기하학적 구조 (surface geometry itself)를 바꾸지 않음.
 - **Position** - 일반적으로 표면에서부터 수직으로 위치를 바꿈
 - **Displacement mapping**
 - **Transparency** - glass effects 방법 (no refraction though)
- 하드웨어에서의 텍스처 맵핑은 하드웨어에서 ⁵⁴텍스처 맵핑에 일치하는 라이팅 계산을 해줘야 가능함

Modeling

- Levels
 - 2차원 하드웨어 - pixels
 - 3차원 하드웨어 - filled triangles with textures and Gouraud shading
 - 저수준 3차원 그래픽 라이브러리 - same plus Phong shading, 변환 (transformations), 투영 (projections), 클리핑 (clipping)
 - 그래픽 엔진 - levels of detail, curved primitives, etc
 - 모델링 패키지 도구 - solid modeling, CSG, fractals, parametric surfaces, etc.

55

Polygonal Representations

- Polygon
 - Gouraud 나 Phong shading 을 사용하여 smooth한 표면
 - 텍스처 맵핑을 사용해서 일부 상세한 표현 가능
- 폴리곤 설계의 어려움
 - Manual
 - 3차원 스캐닝
 - 어떻게 점들을 연결할 지
 - 인터랙티브한 생성
 - 기본 요소 (구, 입방체, 등), extrusions, sweeping volumes
 - Mesh generation
 - Density dependent on curvature
 - Problems with crossing boundaries
 - 고수준 기하요소 생성
 - Constructive solid geometry (CSG), Extrusions, Curves and surfaces

56

Polygonal representations

□ Level of Detail (LOD)

- 사용하는 이유
 - 메시의 단순화 (mesh simplification)
 - 거리에 따라 다른 복잡한 구조를 사용함 (level of detail approximation)
 - 인터넷으로 프로그래시브한 전송 (progressive transmission)
- 문제점
 - 어떤 정점을 버려야 할 지
 - 어떻게 부드러운 표면으로 만들지 (geomorph)
 - 픽셀 수준의 몰핑 (morphing)을 사용함
 - 어떻게 저장을 줄일지
 - 모델의 일부를 선택적으로 개선할 지
- 기법
 - Resampling
 - Edge collapse/edge swaps
 - Vertex removal and retriangulation

57

Landscape/Terrains

□ Terrains은 특수하게 처리해야 함

- 연결된 것 (continuous)
- 매우 커서 물체 클리핑이 필요하지 않는 것 (no object clipping)
- LOD가 좀 더 어려움

□ Height fields

- 픽셀이 높낮이 정보인 2차원 비트맵(bitmap) 사용
- 쉽게 그릴 수 있음
- 쉽게 삼각형 표현으로 변환시킬 수 있음
- 별개의 색깔맵 (color map)이나 높낮이 정보 (height field)로 생성

□ Fractal landscape generation

- 모델링 패키지 지원

□ Level of Detail (LOD)

- Triangle binary tree 를 사용
- LOD시 갑작스럽게 사라지는 peaks나 valleys의 문제가 있음
 - 다른 LOD에서 높이 간의 보간이 필요함⁵⁸
- 보다 나은 방법으로, 주요 특징에 기반한 tree를 구성함

Additional issues

□ 그림자 (shadows)

- 빛과 물체 하나 하나씩에 관련된 계산이 요구됨
- 지면에서만 사용 (지면을 하나의 별개 물체로 취급함)
- Shadow volumes
- Shadow z-buffers

□ 투명 (transparency)

- Back to front drawing order

□ 반사 (reflections)

- 반사맵 (reflection maps)
- 스텐실 버퍼를 사용하여 더블 렌더링 (double rendering)

□ 셰이더 (shaders)

- HLSL, GLSL, Cg
- 프로그램으로 버텍스나 픽셀의 내용을 변경

59

Efficiency

□ 아직도 게임 그래픽을 효과적으로 만들기는 어려움

□ 때문에, 주의 깊은 설계가 요구됨

- Level of detail (LOD) techniques
- 클리핑 (clipping)
- 포탈 기법 (portal techniques)
- 레벨 디자인 (level design)
- 충돌 검사 (collision detection)

60

Reference

- ▣ <http://www.evl.uic.edu/spiff/class/cs426/>
- ▣ <http://dis.dankook.ac.kr/lectures/cg12/>