2024-07-26-백민호

요약

1. CG-Images
    - Image vs Digital Image
      : finite, discrete, pixel(picture element)
    - Raster Graphic
      : bitmap 표현, 2d grid map, array
        - 실제로 들어오는 빛을 추정
        - pixel은 전체를 대표하는 것이 아니라 중앙의 채널 값을 나타낸다
    - Vector Graphic
      : 점, 선, 도형 등 기초 기하와 수학을 이용해 표현한다
        - output 과정에서 자동으로 rasterized 되어 보여질 수 있음
        - 점, 선, polygon 등으로 구성된 3d vector graphic을 input 했을 때 화면 상에 띄울 수 있는 raster images로 바꿀 수 있어야 함!
    - capture image(camera), represent image(bin), display image(monitor)
    - Image 2d array는 Main Memory(RAM)의 Frame Buffer 구역에서 관리
    - GPU Memory가 copy하여 사용한다
2. CG-Graphic-System
    - Image Formation
      : 공간 상 점을 2d plane에 projection 할 때 어디에 위치해 있을까?
        - 광학을 시뮬레이션 하면서 이미지를 생성하는 rendering
        - Light sources, Objects, Camera
    - Graphics Systems
        - Global illumination(Ray Tracing) vs Local/direct illumination
        - local은 object 간 빛 반사, 굴절, 그림자 등 표현 X --> 추정하기
    - Raster Pipeline
        - object가 일련의 과정을 통해 처리되는 것, 한번에 여러 units이 동시에 여러 objects 처리
        - Local/direct illumination은 object마다 독립적이기 때문에 동시에 병렬 처리가 가능하다
            1. host는 main memory의 vertex data(buffer)를 gpu memory로 복사하여 옮긴다. 만약 vertex 정보가 바뀐다면 gpu-side data update

2. 4x4 matrix transformation을 통해 vertex의 3d point info를 local 좌표에서 global 좌표, 관측자 좌표로 positioning, projection
   (Vertex Shader)
3. 이후 vertices --> geometric objects로 collect
4. 시야각을 벗어난 부분을 처리하는 방법
   : clipping volume, view volume
5. Rasterization : clipped 되지 않은 vector graphic 꼴의 object를 potential pixels(called fragments)로 변환 하는 과정
6. 이후 Fragment Shader
7. fragments attributes는 주변 vertices와 fragments에 의해 보간
8. fragments 각각 color, depth 등 적용 후 frame buffers 생성

- Tessellation Shaders
  - 고 수준의 기하 구조를 단순한 삼각형 같은 primitive로 쪼개는 과정
  - Tessellation Control Shader : tessellation 인자들을 결정
  - Tessellation Primitive Generation : tessellation 수행(primitive 변환)
  - Tessellation Evaluation Shader : tessellation 후 생성된 vertices 처리
  - 새롭게 verices들이 생성되는 것 처럼, 단순한 형태를 복잡한 polygon으로 새롭게 만들 수 있다.
- Geometry Shaders(성능이 안좋다)
  - vertex shader(tessellation shader) 이후 processed(if defined)
  - process 당 1개의 primitive 처리
  - primitive 제거하거나 하나의 삼각형을 여러 삼각형으로 확장
  - per-vertex normal을 per-face normal로 변환
- 빛이 많으면 shading 비싸짐(O(N_lights * N_objects)),
  빛마다 Vertex Shader-Rasterizer-Fragment Shader process : Multi-Pass
- Deferred Rendering
  - G-buffer : position, normal, diffuse color, specular, depth 등 light 계산에 필요한 attribute를 저장하는 버퍼
  - Geometry Pass에서 GPU가 MRT 기능을 이용해 G-buffer 값을 저장한다. 이후 종합적으로 Light Pass 시행
  - 한 pixel의 G-buffer를 이용해 light 계산을 하기 때문에 빛 처리 연산이 빨라진다.

3. CG-Geometry
   - vector space : 합과 (scalar)곱으로 이루어진 8가지 공리로 정의된다~

- group : 임의의 operator 에 대해 a, b의 연산이 닫혀있고, 결합, 항등원과 역원을 가질 때 a와 b는 set A에 group이다
  - semigroup은 결합 법칙만 만족할 때, semigroup이다~
  - abelian group(commutative group) : group에 추가로 교환 법칙을 만족할 때
  - ring : addition은 abelian group, multiple은 semigroup
  - ring인데 non-zero element가 (scalar)multiple에서도 abelian group일 때 field!!
- geometry : n-dimension의 objects에 대하여~~
- Point : space location(no size, no shape), point간 distance 정의 필요성 --> scalar
- Scalar : field, set of rules, 혼자서 geometric한 무언가를 할 수 없다
- Vector : 방향만 있음!!(v = P + Q, Q = P + v)
- Euclidean space : vector space + distance measure(최단거리)
- Affine Space : vector space(scalar 포함) + points
  - 기존에는 scalar와 vector 간 연산에서 vector와 point 합을 추가로 정의한다!!!(point - scalar X)
  - 이러한 임의의 vector + point로 vector space를 span 할 수 있다
- Rays : P(k) = Q + kd = Q + k(R - Q)에서 k >= 0 처럼 제한을 둘 때(반직선)
  반직선이 아니라 k가 위아래로 유계가 있을 때는 선분 line segment
- Affine Sum : 아주 제한적인 상황에서 points 간 addition 수행 가능하다?
  P = Q + k(R - Q) 에서 (1-α)Q + αR 꼴로 바꾸는 것~ α 합이 1인 조건으로 n개의 포인트도 sum 연산을 정의할 수 있게 된다
- α_i 값들이 모두 양수이면서 sum(α) = 1일 때 convex hull, 각 α 값들이 1/n 일 때 무게 중심 : barycentric
- modern graphics는 triangles로 이루어진 mesh에 최적화 되어있음!! (point랑 line도!!)
  - triangle은 언제나 flat하기 때문이다!!
  - N개의 triangle --> vertex 3개 위치를 저장하거나 위치 저장 보다는 topology를 이용해서 위치는 계속 바뀌는데 연결 관계는 동일하거나 그런 느낌쓰
  - 인덱스 버퍼를 만들어서 topology를 geometry와 분리시키자, 인덱스가 vertex 위치를 가리키기만 하면 된다~

4. Transformation
   - linearly independent(선형 독립)과 dimension, basis(기저, not unique) 조합은 unique
   - representation(좌표)는 basis의 조합 {α_i}로 나타낸다

- coordiante-free geometry : point의 좌표 없이 표현(예) 유클리드 기하 -> 길이로)
- coordinate system은 vector만을 이용한다, v = 2v_1 + 3v_2 - 5v_3
  - 그래서 고정된 위치는 없지만 결과값은 같게 나온다
  - 위치(point)에 대한 concept을 넣어야 하기 때문에 frame 등장
- 우리가 생각하는 좌표계는 frame, point와 vector로 affine space를 만들 수 있다~
  - 기준이 되는 P_0를 잡자, P = P_0 + 2v_1 + 3v_2 - 5v_3
  - P = [α1, α2, α3, w] * [v1, v2, v3, P0]
- 4-d homogeneous coordinate(HC) representation p = [x', y', z', w]
  - 3D Cartesian 좌표계의 vector와 points를 표현한다
  - vector(w = 0), point(w = 1), else x', y', z'을 w로 나누어진 point
  - w를 1로 바꾸면서 차원을 축소시킬 수 있다(사영시킨다)
- Affine Transformation
  - affine space를 다룬다. points, lines, planes는 보존된다
    : 직선이 곡선이나 선분으로 바뀌지 않는다
  - linear transformation은 frames(P0)을 바꾸는 것과 동일하다
  - 4 by 3 matrix --> translation, rotation, scaling, shear 총 12개 dof
  - 선분을 transform 할 때는 end point만 이동시킨다
- Translation
  - p' = p + d, 3 dof, p' = Tp 꼴로 바꾸기
- 동일 방법으로 Scaling, Rotation(어떤 축 v를 중심으로)
  - 회전의 경우 x, y, z 축 θ로 decomposed 가능하다 벗!! 회전하다가 두 축이 겹치게 되는 경우, gimbal lock을 유발할 수 있으니 Euler angle을 쓰지 말자
- Inverse Transformation
  - 역행렬은 정확하지도 않고 비싸기 때문에 기하학적으로 역을 구하고 matrix로 표현하자

5. Viewing
   - Objects, Viewer(CoP, eyes, camera ...), Projection surface, Projectors
   - Parallel Projection
     - COP -> inf, projectors가 서로 평행하게, 이때 COP는 point가 아니라 방향(DOP)
     - projectors가 projection surface에 orthognal 할 때 --> orthographic projection
   - Perspective Viewing
     - diminution : 물체들이 viewer에서 멀어질 수록 작아지는 거~
     - 줄어드는 정도에 따라 depth를 추론할 수 있다, 정확도 보단 natural-looking

- Backward Approach
  - 주로 ray tracing, 빛이 오는 반대 방향으로 ray를 쏘는 것
  - 카메라 sensor에서 object 방향으로 쏘면 backward, image plane에 pixel 만듦
- Forward Approach(pipeline approach)
  - 물체에서 바로 image plane에 projection하는 것, matrix의 multiplication 꼴
  - modeling, view(camera), projection, viewport 순서로 transformation
  - frame을 각각 바꾼다(object > world > camera)
- Moving Camera : Building LookAt Matrix
  - Model-view duality
    - camera를 움직일 것인가 model을 움직일 것인가??(결과는 같다)
    - object를 camera 반대 방향으로 움직일 것이다
    - camera는 origin으로
  - LookAt Method
    - 위치와 방향 회전을 다 지정해주는 방법, mat4 lookat(eye, at, up)
    - eye에서 at 쪽으로(view vector) 각도는 up, world space 기준!!
    - up 벡터는 그 카메라 막 옆으로 돌려서 찍고 그런 거
    - camera frame(space) 추론 가능!!
      - n = norm(eye - at)
      - u = norm(up X n)
      - v = norm(n X u), basis vector 닌깐,,,
    - 이걸로 change of frame 가능!! world frame -> camera frame
    - 4 X 4 LookAt matrix
    - orthonormal basis의 transpose는 (R^T)^-1 == (R^T)^T
    - 따라서 Ra = b 꼴로 world-coordinate에서 a를 eye-coordinate의 b로 바꿀 수 있다!! camera frame으로

6. Projection
   - Coordinate System
     - RHS 이용하기
     - Normalized Device Coordiante(NDC)
       - camera space에서 projection이 끝난 뒤에 쓴다
       - RHS -> LHS z 값이 반대로 이동한다
       - depth test를 위해 직관적으로 이해하기 위해 방향을 맞춘 것이다
   - Orthographic projection(Matrix)
     - COP를 무한대로 보내면 된다!! -> parallel projection

    - (x, y, z)를 xy plane에 투영시킬 때 -> (x, y, 0) 그냥 depth만 0으로 바꾸면 된다
    - 3d graphic api에서는 z를 남겨야 한다, 3d를 유지해야하기 때문!!
    - ndc에 있는 view volume normalization 추가(물체들이 있을 수 있는 영역, cube를 설정한다)
    - 2d plane projection에서 view volume normalization으로 바뀌게 된다(z != 0)
    - projection이 일어난 뒤에도 4d homogeneous coordinate
    - depth를 [0, 1]로 norm 해서 나중에 fragment 들어왔을 때 남길지 말지 결정
    - clipping도 쉬워진다!
    - translation + scaling
        - view volume의 중심을 0, 0, 0으로 맞춘다
        - (1,1,1), (-1, -1, -1)의 점을 갖도록(한 변의 길이가 2가 되도록) scaling 한다
- Perspective projection
    - simple perspective projection
    - (x, y, z)를 원점 방향(COP)으로 옮겼을 때 intersection
        - z' = d일 때 나머지 x'와 y'을 비례식으로 구하기
    - 우선 symmetric한 경우
    - simple perspective proj + VVN(요것도 depth가 바뀐다)
        - 이번에는 (left, bottom, -near)와 (right, top, -near)(not -far)을 사용한다
        - near 부분과 far 부분의 사각형 크기가 다르기 때문!! -far 부분은??
        - l, b, r, t를 near plane에서 체크하기
            - x = -+z, y = -+z 로 바꾸기 위해 scaling, z는 -far 부터 -near 까지
            - r/n 기울기를 +-1인 형태로 x와 y를 바꾼다
            - z, depth를 -near에서 -1, -far에서 +1로 매핑이 되어야 한다(뒤집힌다)
        - l, r, t, b를 측정하기가 어렵기 때문에 field of view(fovy, fovx, 화각)와 aspect_ratio(width/height of sensor)를 이용해서 측정한다~
    - Non-Symmetric Perspective Projection(View가 1개 이상 또는 기울어질 때 등)
    - 눈이 두 개이니까 VR도 이거임!!
    - non-sym를 shear 해서 sym으로 바꾸는 것이 중요!!

- near plane과 far plane의 중심점 (l+r)/2와 (t+b)/2를 0,0으로 맞추어야 한다!