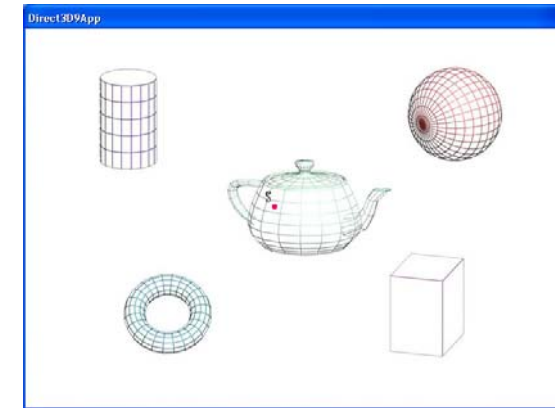


Picking

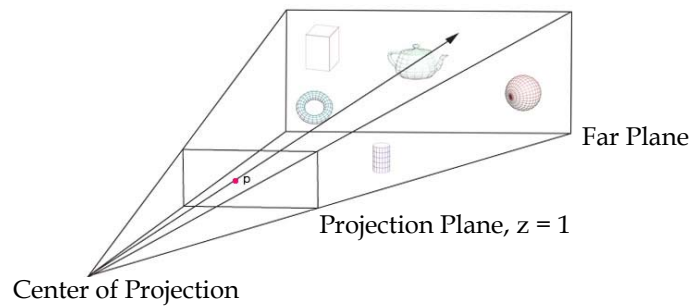
305890
2007년 봄학기
5/30/2007
박경신

Picking



사용자가 주전자를 선택하였을 경우 - 스크린 좌표를 이용하여 선택한 물체를 알아내는 것을 **픽킹(picking)**이라 부름

Picking



원점에서 출발하여 p 를 통과하는 광선이 한 물체와 교차.
투영창의 p 는 화면의 s 에 대응.

Picking

- Screen to Projection Window Transformation
 - 스크린 포인트 s 를 통해 투영창에 대응되는 포인트 p 를 계산
- Computing the Picking Ray
 - 원점에서 출발하여 p 를 통과하여 발사되는 픽킹 광선을 계산
- Transforming Rays
 - 픽킹 광선과 모델을 동일한 공간으로 변환
- Ray-Object Intersections
 - 픽킹 광선과 모델의 교차 계산. 교차된 물체는 선택된 스크린 물체와 대응
- Picking 예제

Picking

□ Picking

- 사용자가 screen point $s=(x, y)$ 를 가리켰을 때, 위치 s 에 해당하는 물체를 알아내는 기법
- 게임 제작에서 매우 중요한 요소임
- 예) 물체를 click해서 작동 방식 선택, 적군을 click해서 무기를 발사, 아이템을 click해서 집는 동작 등등.

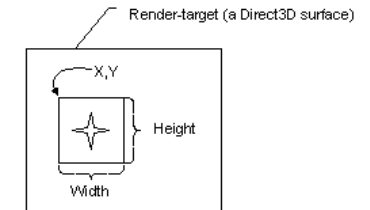
□ Picking 기법의 구현 절차

1. Screen point, s 에 대응되는 투영창 ($z=1$)에서의 점 p 를 계산
2. 원점에서 p 로 향하는 **picking ray** 계산
3. Picking ray와 model을 동일 공간으로 **변환**
4. Picking ray와 **교차**하는 물체를 알아냄

Screen to Projection Window Transformation

□ Viewport transformation matrix

$$\begin{bmatrix} \frac{Width}{2} & 0 & 0 & 0 \\ 0 & -\frac{Height}{2} & 0 & 0 \\ 0 & 0 & MaxZ - MinZ & 0 \\ X + \frac{Width}{2} & Y + \frac{Height}{2} & MinZ & 1 \end{bmatrix}$$



```
typedef struct _D3DVIEWPORT9 {
    DWORD X, Y; // the upper-left corner (in pixel coordinates)
    DWORD Width, Height; // width, height (in pixel)
    float MinZ, MaxZ; // range of depth value - 일반적으로 0.0 또는 1.0
} D3DVIEWPORT9;
```

Screen to Projection Window Transformation

- 투영창의 점 $p=(p_x, p_y, p_z)$ 에 Viewport 변환행렬을 적용하면 screen point $s=(s_x, s_y)$ 가 얻어짐

$$s_x = p_x \left(\frac{Width}{2} \right) + X + \frac{Width}{2}$$

$$s_y = -p_y \left(\frac{Height}{2} \right) + Y + \frac{Height}{2}$$

- 주어진 스크린 포인트 s 를 이용해 p 를 구함

$$p_x = \frac{2s_x - 2X - Width}{Width}$$

$$p_y = \frac{-2s_y + 2Y + Height}{Height}$$

Screen to Projection Window Transformation

- $X = Y = 0$ 을 가정하면 투영창은 평면 $z=1$ 과 일치

$$p_x = \left(\frac{2s_x}{Width} \right) - 1, p_y = \left(\frac{-2s_y}{Height} \right) + 1, p_z = 1$$

- 투영 행렬 (projection matrix) P 를 사용한 투영창의 배율 변경

- P_{00} 과 P_{11} 항목이 x, y 좌표의 배율을 변경하므로

$$p_x = \left(\frac{2x}{viewportWidth} - 1 \right) \left(\frac{1}{P_{00}} \right), p_y = \left(\frac{-2y}{viewportHeight} + 1 \right) \left(\frac{1}{P_{11}} \right), p_z = 1$$

Computing the Picking Ray

□ Picking Ray

- Ray: 위치와 방향
- $p(t) = p_0 + tu, (t \geq 0)$

```
struct Ray {
    D3DXVECTOR3 _origin;
    D3DXVECTOR3 _direction;
};
```

□ Picking Ray

- $p = (p_x, p_y, 1)$ 를 ray가 지나가는 투영창의 점이라고 하면
- $p_0 = (0, 0, 0)$ 이므로, 방향 벡터 $u = p - p_0 = p$
- Ray = $p_0 + tu$

Computing the Picking Ray

```
d3d::Ray CalcPickingRay(int x, int y) // screen point (sx, sy)
{
    float px = 0.0f;
    float py = 0.0f;

    D3DVIEWPORT9 vp;
    Device->GetViewport(&vp);

    D3DXMATRIX proj;
    Device->GetTransform(D3DTS_PROJECTION, &proj);
    px = (((2.0f * x) / vp.Width) - 1.0f) / proj(0, 0);
    py = (((-2.0f * y) / vp.Height) + 1.0f) / proj(1, 1);

    d3d::Ray ray;
    ray._origin = D3DXVECTOR3(0.0f, 0.0f, 0.0f);
    ray._direction = D3DXVECTOR3(px, py, 1.0f);

    return ray; // return ray (origin, direction)
}
```

Transforming Ray

□ Picking Ray

- Ray와 물체의 교차 검사(intersection test)를 위해서는 동일 좌표 시스템으로 바꾸어야 함
- View space에서의 picking ray를 world space로 바꾼다
 - Picking Ray의 점 p_0 와 방향 u 를 view 변환 행렬의 역행렬로 변환

```
void TransformRay (d3d::Ray* ray, D3DXMATRIX* T) {
    // transform the ray's origin, w = 1
    D3DXVec3TransformCoord(&ray->_origin, &ray->_origin, T);
    // transform the ray's direction, w = 0
    D3DXVec3TransformNormal(&ray->_direction, &ray->_direction, T);
    // normalize the direction
    D3DXVec3Normalize(&ray->_direction, &ray->_direction);
}
```

D3DXVec3TransformCoord: (x, y, z, w=1)
D3DXVec3TransformNormal: (x, y, z, w=0)

Ray-Object Intersections

□ Intersection Test

- 방법 1: 물체를 이루는 모든 triangle들에 차례로 ray intersection test함. 그러나 계산량이 많음.
- 방법 2: 각 물체의 경계 구체(bounding volume)와 ray intersection test함. 정확성은 다소 떨어지지만 효율적인 방법임.
- 다수의 물체와 교차되는 경우에는 camera와 가까운 물체를 선택함.

Ray-Object Intersections

□ Sphere-Ray Intersection Test

■ Sphere의 중심점 c 와 반지름 r

□ 점 p 가 sphere상에 있을 조건: $\|p - c\| - r = 0$

□ Ray $p(t) = p_0 + tu$ 가 sphere와 intersect하는 지를 검사

$$\|p - c\| - r = 0$$

$$\Rightarrow \|p(t) - c\| - r = 0$$

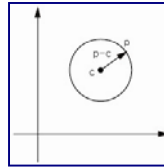
$$\Rightarrow \|p_0 + tu - c\| - r = 0$$

$$\Rightarrow At^2 + Bt + C = 0$$

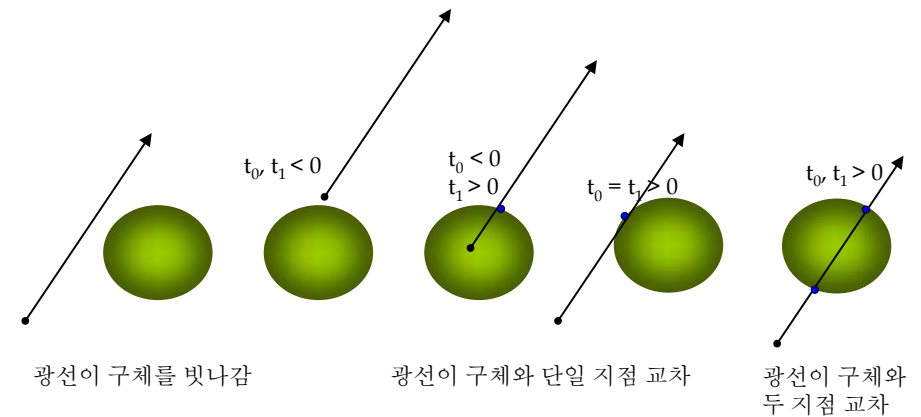
$$A = u \cdot u, B = 2(u \cdot (p_0 - c)), C = (p_0 - c) \cdot (p_0 - c) - r^2$$

$$t = \frac{-B \pm \sqrt{B^2 - 4C}}{2}$$

=> t 의 두 개의 값 중 양수의 경우가 intersection을 의미함



Ray-Object Intersections



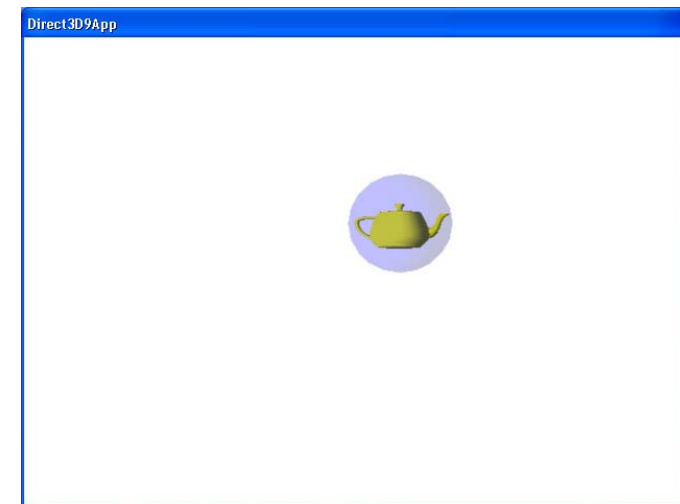
Ray-Object Intersections

```

struct BoundingSphere {
    D3DVECTOR3 _center;
    float      _radius;
};

bool RaySphereIntTest (d3d::Ray* ray, d3d::BoundingSphere* sphere) {
    D3DXVECTOR3 v = ray->_origin - sphere->_center;
    float b = 2.0f * D3DXVec3Dot(&ray->_direction, &v);
    float c = D3DXVec3Dot(&v, &v) - (sphere->_radius * sphere->_radius);
    float discriminant = (b*b) - (4.0f * c); // discriminant
    if (discriminant < 0.0f) // test for imaginary number
        return false;
    discriminant = sqrtf(discriminant);
    float t0 = (-b + discriminant) / 2.0f;
    float t1 = (-b - discriminant) / 2.0f;
    if (t0 >= 0.0f || t1 >= 0.0f) // if a solution is >= 0, then we intersected the sphere
        return true;
    return false;
}
    
```

Example: Picking



Example: Picking

```
ID3DXMesh* Teapot = 0;    ID3DXMesh* Sphere = 0;
D3DXMATRIX World;
d3d::BoundingSphere BSphere;

bool Setup() {
    D3DXCreateTeapot(Device, &Teapot, 0);
    // Compute the bounding sphere
    BYTE * v = 0;
    Teapot->LockVertexBuffer(0, (void**) &v);
    D3DXComputeBoundingSphere((D3DXVECTOR3*)v, Teapot->GetNumVertices(),
        D3DXGetFVFVertexSize(Teapot->GetFVF()), &BSphere._center, &BSphere._radius);
    Teapot->UnlockVertexBuffer();
    // Build a sphere mesh that describes teapot's bounding sphere
    D3DXCreateSphere(Device, BSphere._radius, 20, 20, &Sphere, 0);

    // light
    // view matrix
    // projection matrix

    return true;
}
```

Example: Picking

```
LRESULT CALLBACK d3d::WndProc(...) {
    switch (_msg) {
        case WM_LBUTTONDOWN:
            // compute the ray in view space given the clicked screen point
            d3d::Ray ray = CalcPickingRay(LWORD(IParam), HIWORD(IParam));
            // transform the ray to world space
            D3DXMATRIX view;
            Device->GetTransform(D3DTS_VIEW, &view);
            D3DXMATRIX viewInverse;
            D3DXMatrixInverse(&viewInverse, 0, &view);
            TransformRay(&ray, &viewInverse);
            if (RaySphereIntersectTest(&ray, &BSphere)) // test for a hit
                ::MessageBox(0, "Hit!", "HIT", 0);
            break;
    }
}
```

Example: Picking

```
bool Display(float timeDelta) {
    if (Device) {
        // update teapot
        static float r=0.0f, v=1.0f, angle=0.0f;
        D3DXMatrixTranslation(&World, cosf(angle)*r, sinf(angle)*r, 10.0f);
        BSphere._center = D3DXVECTOR3(cosf(angle)*r, sinf(angle)*r, 10.0f);
        r += v * timeDelta;
        if (r >= 8.0f) v = -v; // reverse direction
        if (r <= -8.0f) v = -v; // reverse direction
        angle += 1.0f * D3DX_PI * timeDelta;
        if (angle >= D3DX_PI * 2.0f) angle = 0.0f;
        // render
        Device->Clear(0, 0, D3DCLEAR_TARGET | D3DCLEAR_ZBUFFER, 0x00000000, 1.0f, 0);
        Device->BeginScene();

        // render teapot
        Device->SetTransform(D3DTS_WORLD, &World);
        Device->SetMaterial(d3d::YELLOW_MTRL);
        Device->DrawSubset(0);
    }
}
```

Example: Picking

```
        // render bounding sphere with alpha blending
        Device->SetRenderState(D3DRS_ALPHABLENDENABLE, true);
        Device->SetRenderState(D3DRS_SRCBLEND, D3DBLEND_SRCALPHA);
        Device->SetRenderState(D3DRS_DESTBLEND, D3DBLEND_INVSRCALPHA);
        D3DMATERIAL9 blue = d3d::BLUE_MTRL;
        blue.Diffuse.a = 0.25f; // 25% opacity
        Device->SetMaterial(&blue);
        Device->DrawSubset(0);
        Device->SetRenderState(D3DRS_ALPHABLENDENABLE, false);

        Device->EndScene();
        Device->Present(0, 0, 0, 0);
    }
    return true;
}
```