It’s All Done with Mirrors

GPU Gems Showcase
Kevin Bjorke, NVIDIA
GDC2004
Shiny Is Good

- Making objects shiny via reflection maps is one of the cheapest ways to add visual complexity.
- If there’s no bump map, simple cases can be done in the vertex shader.
Polishing “Shiny”

• All specular light is actually scattered shiny reflection

• Can we use reflections in place of specular light?

• Can we make reflections behave more like… lights?
Localizing Reflections

- Finite-Radius cube maps
- Adding reflective primitive objects
- Both are forms of very limited ray tracing
The usual way to use cube maps

- We assume the cube is infinitely far away (like the sky)
- We calculate the reflection vector and look up in the cube map

\[ R = 2(N \cdot V)N - V \]
Infinite Map Characteristics

- Since $R$ is known at each vertex, we can just use a vertex shader (barring bump maps)
- Every point is at the “center” of the map
- Translation of object w/ camera has no effect
Finite-Radius Maps

- If the map has a specific radius and center location, then:
- Only one point is at the “center” of the map
Every Reflection is “Local”

• Translating the object (or the map) means a change in the reflection
• We need just a little pixel-shader math (ray-sphere intersection test)
Finite Reflections at Home

• “simplerFiniteReflection.fx” is part of the FX Composer release
  – Lets us pick a map center and radius
  – Applies bump-normal map
• **GPU Gems** version describes map rotations too
Shader Connectors

/* data from application vertex buffer */
struct appdata {
    half3 Position : POSITION;
    half4 UV : TEXCOORD0;
    half4 Normal : NORMAL;
    half4 Tangent : TANGENT0;
    half4 Binormal : BINORMAL0;
};

/* data passed from vertex shader to pixel shader */
struct vertexOutput {
    half4 HPosition : POSITION;
    half4 TexCoord : TEXCOORD0;
    half3 UserNormal : TEXCOORD1; // "User" Space == Env Map Coords
    half3 WorldEyeVec : TEXCOORD2;
    half3 UserTangent : TEXCOORD3;
    half3 UserBinorm : TEXCOORD4;
    half3 UserEyeVec : TEXCOORD5;
    half3 UserPos : TEXCOORD6;
};
vertexOutput mainVS(appdata IN) {
    vertexOutput OUT;
    half4x4 userXf;
    half ir = 1.0/ReflScale;
    userXf[0] = half4(ir,0,0,ReflCenter.x); // These constant xforms could be built by the Direct X Virtual Machine
    userXf[1] = half4(0,ir,0,ReflCenter.y);
    userXf[2] = half4(0,0,ir,ReflCenter.z);
    userXf[3] = half4(0,0,0,1);
    half4x4 userITXf;
    userITXf[0] = half4(ReflScale,0,0,0);
    userITXf[1] = half4(0,ReflScale,0,0);
    userITXf[2] = half4(0,0,ReflScale,0);
    userITXf[3] = half4(-ReflCenter.xyz,1);
    half4 Po = half4(IN.Position.xyz,1.0); half4 Pw = mul(Po, World);
    OUT.TexCoord = IN.UV;
    half3 WorldEyePos = ViewIT[3].xyz;
    half4 UserEyePos = mul(half4(WorldEyePos,1.0),userXf);
    half4 hpos = mul(Po, WorldViewProj);
    half4 Pw = mul(Pw,userXf);
    half4 Nw = mul(IN.Normal, WorldIT);
    half4 Tw = mul(IN.Tangent, WorldIT);
    half4 Bw = mul(IN.Binormal, WorldIT);
    OUT.UserEyeVec = (UserEyePos - Pw.xyz);
    OUT.WorldEyeVec = normalize(WorldEyePos - Pw.xyz);
    OUT.UserNormal = mul(Nw,userITXf).xyz;
    OUT.UserTangent = mul(Tw,userITXf).xyz;
    OUT.UserBinorm = mul(Bw,userITXf).xyz;
    OUT.UserPos = mul(Pw,userXf).xyz;
    OUT.HPosition = hpos;
    return OUT;
}
half4 mainPS(vertexOutput IN) : COLOR
{
    half3 Vu = normalize(IN.UserEyeVec);
    half3 Nu = normalize(IN.UserNormal);
    half3 Tu = normalize(IN.UserTangent);
    half3 Bu = normalize(IN.UserBinorm);
    half3 bumps = Bumpiness * (tex2D(NormalSampler,IN.TexCoord.xy).xyz-(0.5).xxx);
    half3 Nb = Nu + (bumps.x * Tu + bumps.y * Bu);
    Nb = normalize(Nb);   // expressed in user-coord space
    half vdn = dot(Vu,Nu);
    half fres =  KrMin + (Kr-KrMin) * pow((1.0-abs(vdn)),FresExp);
    half3 reflVect = -normalize(reflect(Vu,Nb));  // yes, normalize in this instance!
    // we are using the coord sys of the reflection, so to simplify things we assume (radius == 1)
    half b = -2.0 * dot(reflVect,IN.UserPos);
    half c = dot(IN.UserPos,IN.UserPos) - 1.0;
    half discrim = b*b - 4.0*c;
    bool hasIntersects = false;
    half nearT = 0;
    half3 reflColor = half3(1,0,0);  // red
    if (discrim > 0) {
        discrim = sqrt(discrim);
        nearT = -(discrim-b)/2.0;
        hasIntersects=true;
        if (nearT <= 0.0001) {
            nearT = (discrim - b)/2.0;
            hasIntersects = (nearT>0.0001);
        }
    }
    if (hasIntersects) {
        reflVect = -IN.UserPos + nearT*reflVect;
        reflColor = fres * texCUBE(EnvSampler,reflVect).xyz;
    }
    half3 result = SurfColor * reflColor.xyz;
    return half4(result,1);
}
Finite Results

Small Environment

Large Environment
Possible Variations

• Refraction
• Integrate Diffuse Terms
• Distance Falloff
• Other shapes (e.g., a cube instead of a sphere)
• Fresnel Terms
Reflection Cards

- Like finite-radius environment maps, we can do limited raytracing against simple, known objects in the scene – for example, a (textured) plane....
“reflCard.fx” in GDC Web Pkg

• Available soon at
  http://developer.nvidia.com
vertexOutput sa2VS(appdata IN, uniform REAL4x4 CardXf)
{
    vertexOutput OUT = (vertexOutput)0;
    REAL3 Nw = mul(IN.Normal,WorldITXf).xyz;
    REAL4 Po = REAL4(IN.Position.xyz,(REAL)1.0);  // obj coordinates
    REAL4 Pw = mul(Po,WorldXf);  // world coords
    REAL4 Pc = mul(Pw,CardXf);  // card coords
    OUT.CPos = Pc.xyz;  // card coords
    REAL4 Ec = mul(float4(ViewIXf[3].xyz,1),CardXf);
    OUT.View = normalize(Ec.xyz - Pc.xyz);  // card coords
    REAL3x3 rx = REAL3x3(CardXf[0].xyz,CardXf[1].xyz,CardXf[2].xyz);
    OUT.Normal = mul(Nw,rx);  // card xf
    OUT.HPosition = mul(Po,WorldViewProjXf);// screen clipspace coords
    OUT.UV = IN.UV.xy;
    return OUT;
}
REAL4 reflCardPS(vertexOutput IN) : COLOR
{
    REAL3 Nn = NORM(IN.Normal);
    REAL3 Vn = NORM(IN.View);
    REAL3 refl = reflect(Vn,Nn);
    REAL t = (IN.CPos.z/refl.z);
    REAL3 result = 0;  // or potentially, set to border color
    if (t > 0.0) {
        REAL3 zIntersect = IN.CPos - refl*t;
        REAL2 cardUV = (zIntersect.xy /
            REAL2(CardWidth,CardHeight)) +
            REAL2(0.5,0.5);
        result = tex2D(CardSampler,cardUV).xyz;
    }
    return REAL4(result,1);
}
Card Result
Possible Enhancements

- Distance Falloff – Make Cards into Light Sources
- Transparency – make arbitrary shapes
- Multiple Cards can obscure one another
- When we miss the card(s), default to a background (potentially finite) Cube Map reflection
- Bump Maps
- Refraction
- Fresnel
Questions?

Thanks!

kbjorke@nvidia.com