Optimized Rendering Techniques Based on Local Cubemaps

ARM

Roberto Lopez Mendez
Senior Software Engineer, ARM

ARM Game Developer Day - London
03/12/2015

Agenda

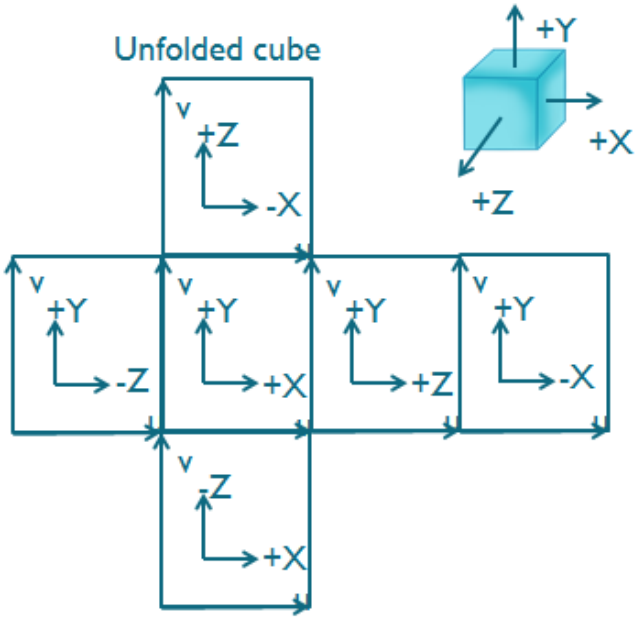
- The concept of local cubemaps
- Optimized rendering techniques based on local cubemaps
  - Reflections
  - Shadows
  - Refractions
- Wrap up

The Concept of Local Cubemaps

Cubemaps

Cubemaps

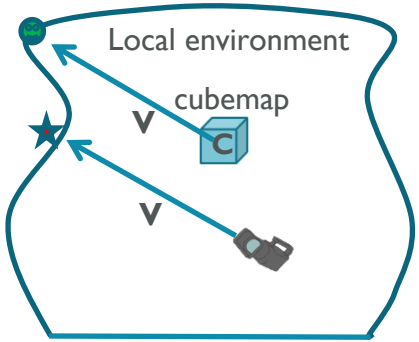
- Hardware accelerated
- No image distortion
- Efficient computation



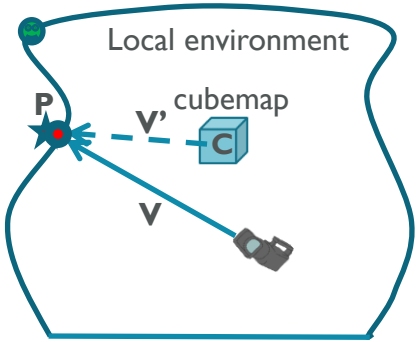

```
float4 col = texCUBE(Cubemap, V);
```


ARM

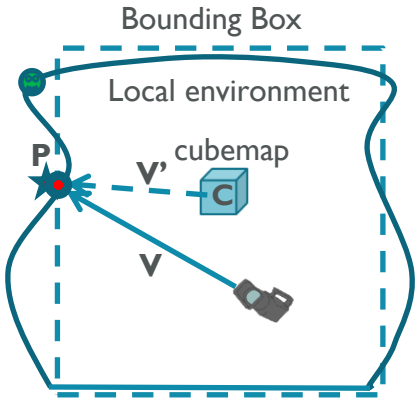
The Concept of Local Cubemaps



If we use the view vector **V** defined in WCS to fetch the texel from the cubemap we will get the smiley face instead of the star.



We need to use a new vector **V'** = **CP** to fetch the correct texel .We need to find the intersection point P of the view vector **V** with the boundaries of the local environment.



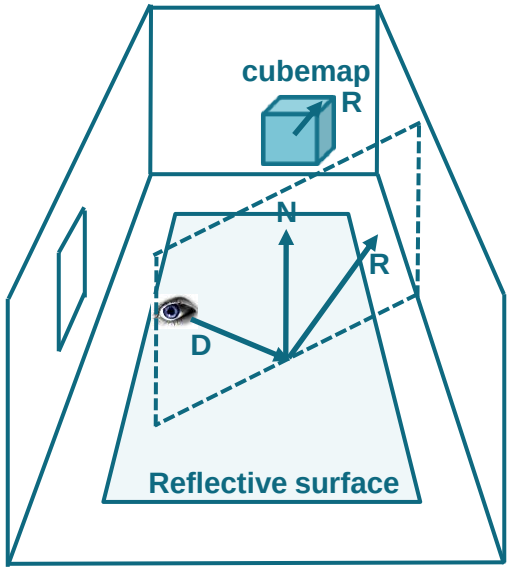
We introduce a proxy geometry to simplify the problem of finding the intersection point **P**.The simplest proxy geometry is the bounding box.

Local Cubemap = Cubemap + Cubemap Position + Scene Bounding Box + Local Correction

ARM

Reflections Based on Local Cubemaps

Reflections with Infinite Cubemaps



Normal N and view vector D are passed to fragment shader from the vertex shader.

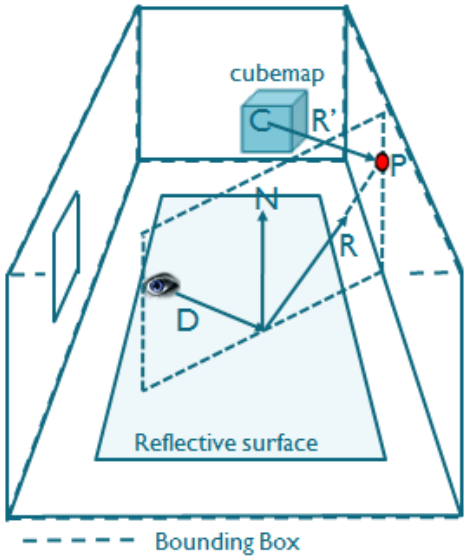
In the fragment shader the texture colour is fetched from the cubemap using the reflected vector R:

float3 R = reflect(D, N);
float4 col = texCUBE(Cubemap, R);

Incorrect Reflections

Reflection generated using a cubemap without any local binding

Local Correction Using a Bounding Box as a Proxy Geometry




```
float3 R = reflect(D, N);

float4 col = texCUBE(Cubemap, R);

Find intersection point P
Find vector R' = CP
float4 col = texCUBE(Cubemap, R');
```

Source code in the ARM Guide for Unity Developers at MaliDeveloper.arm.com

GPU Gems. Chapter 19. Image-Based Lighting. Kevin Bjork, 2004. http://http.developer.nvidia.com/GPUGems/gpugems_ch19.html

Cubemap Environment Mapping. 2010. http://www.gamedev.net/topic/568829-box-projected-cubemap-environment-mapping/?&p=4637262

Image-based Lighting approaches and parallax-corrected cubemap. Sebastien Lagarde. SIGGRAPH 2012. http://seblagarde.wordpress.com/2012/09/29/image-based-lighting-approaches-and-parallax-corrected-cubemap/

Vertex Shader

```
vertexOutput vert(vertexInput input)
{
        vertexOutput output;

        output.tex = input.texcoord;
        // Transform vertex coordinates from local to world.
        float4 vertexWorld = mul(_Object2World, input.vertex);

        // Transform normal to world coordinates.
        float4 normalWorld = mul(float4(input.normal, 0.0), _World2Object);

        // Final vertex output position.
        output.pos = mul(UNITY_MATRIX_MVP,  input.vertex);

        // ---------- Local correction ------------
        output.vertexInWorld = vertexWorld.xyz;
        output.viewDirInWorld = vertexWorld.xyz - _WorldSpaceCameraPos;
        output.normalInWorld = normalWorld.xyz;

        return output;
}
```

Passed as varyings to the fragment shader

Source code in the ARM Guide for Unity Developers

Fragment shader

Calculate reflected vector

Ray-box intersection algorithm

Local corrected reflected vector

```
float4 frag(vertexOutput input) : COLOR
{
        float4 reflColor = float4(1, 1, 0, 0);

        // Find reflected vector in WS.
        float3 viewDirWS = normalize(input.viewDirInWorld);
        float3 normalWS = normalize(input.normalInWorld);
        float3 reflDirWS = reflect(viewDirWS, normalWS);

        // Working in World Coordinate System.
        float3 localPosWS = input.vertexInWorld;
        float3 intersectMaxPointPlanes = (_BBoxMax - localPosWS) / reflDirWS;
        float3 intersectMinPointPlanes = (_BBoxMin - localPosWS) / reflDirWS;
        // Looking only for intersections in the forward direction of the ray.
        float3 largestRayParams = max(intersectMaxPointPlanes, intersectMinPointPlanes);
        // Smallest value of the ray parameters gives us the intersection.
        float distToIntersect = min(min(largestRayParams.x, largestRayParams.y), largestRayParams.z);
        // Find the position of the intersection point.
        float3 intersectPositionWS = localPosWS + reflDirWS * distToIntersect;
        // Get local corrected reflection vector.
        reflDirWS = intersectPositionWS - _EnviCubeMapPos;

        // Lookup the environment reflection texture with the right vector.
        reflColor = texCUBE(_Cube, reflDirWS);
        // Lookup the texture color.
        float4 texColor = tex2D(_MainTex, float2(input.tex));

        return _AmbientColor + texColor * _ReflAmount * reflColor;
}
```

Source code in the ARM Guide for Unity Developers

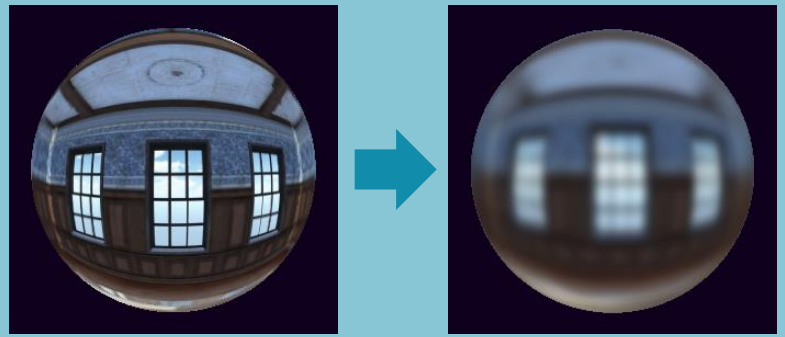
Correct and Incorrect Reflections

Reflection generated after applying the "*local correction*"

Reflection generated without "*local correction*"

Filtering Cubemaps to Achieve Visual Effects

Development Stage

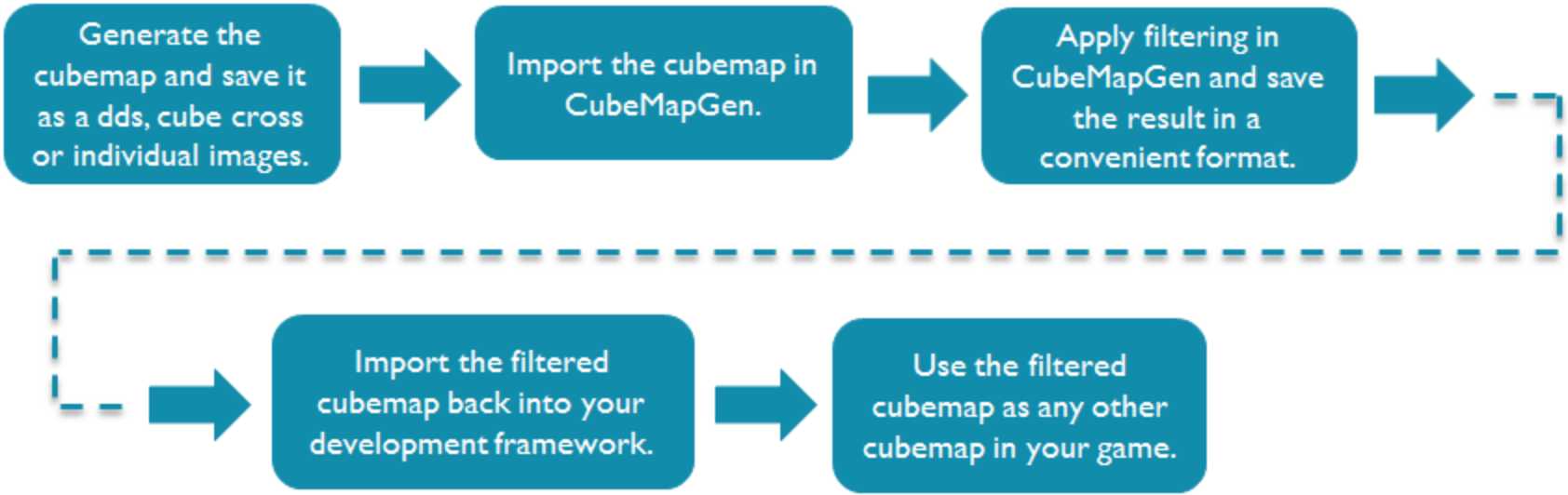


Can take several minutes on a PC

Runtime

Just use the filtered cubemap

Workflow for Offline Cubemap Filtering



CubeMapGen (AMD) - http://developer.amd.com/tools-and-sdks/archive/legacy-cpu-gpu-tools/cubemapgen/

Detailed explanation and source code in the ARM Guide for Unity Developers

Benefits and Limitations

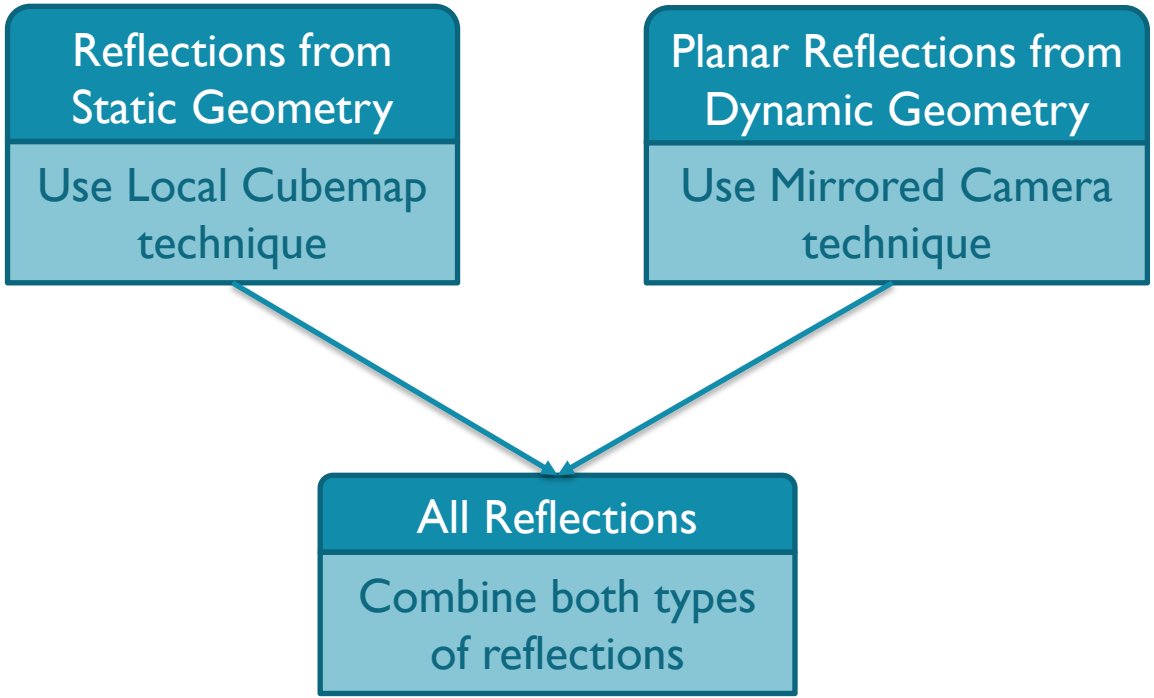
Benefits

1. Simple to implement
2. Very realistic
3. Physically correct
4. High quality of reflections, no pixel flickering/ instability when moving camera
5. Cubemap texture can be compressed
6. Offline filtering effects can be applied which could be very expensive at run time
7. Resource saving technique, important for mobile devices

Limitations

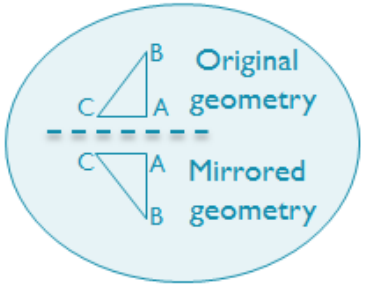
1. Only works well in open plan space with no geometry in the centre where the cubemap will likely be generated
2. Objects in the scene must be close to the proxy geometry for good results
3. Only effective for simulating reflections of static geometry

Handling Reflections of Dynamic Objects

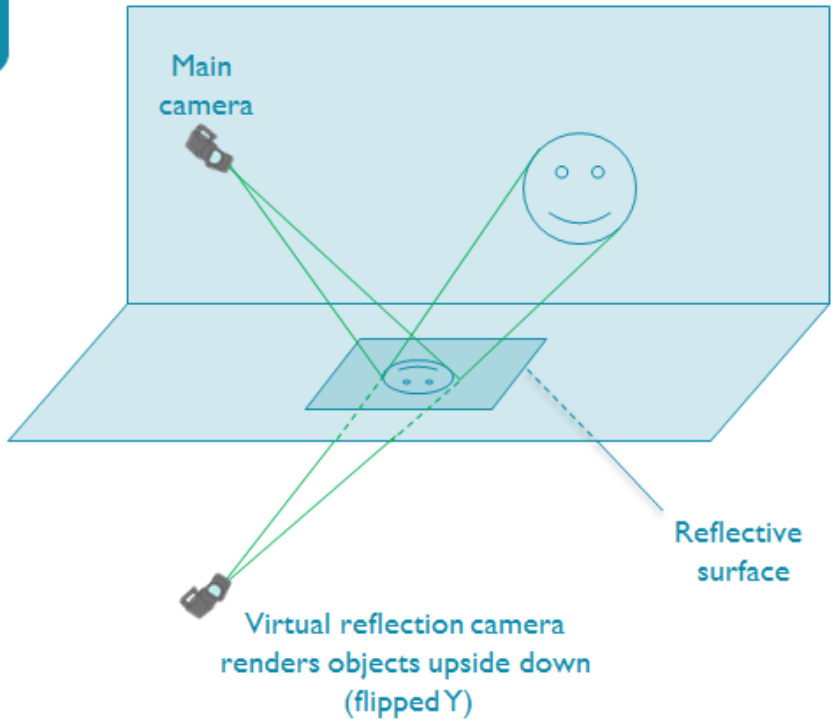


Dynamic Objects Runtime Reflections with a Virtual Camera

Setting the camera upside down affects the winding of the geometry.



We need to reverse geometry when rendering with the reflection camera to fix the winding order.



ARM

The Reflection Matrix

$$R = \begin{bmatrix} 1 - 2n_x^2 & -2n_x n_y & -2n_x n_z \\ -2n_x n_y & 1 - 2n_y^2 & -2n_y n_z \\ -2n_x n_z & -2n_y n_z & 1 - 2n_z^2 \end{bmatrix}$$

$$\begin{aligned} n_x &= planeNormal_x \\ n_y &= planeNormal_y \\ n_z &= planeNormal_z \end{aligned}$$

$$n_w = -dot(planeNormal, normalPos)$$

Rendering Static and Dynamic Reflections - Vertex Shader

```
vertexOutput vert(vertexInput input)
{
        vertexOutput output;

        // Transform vertex coordinates from local to world
        float4 vertexWorld = mul(_Object2World, input.vertex);
        // Transform Normal to world coordinates
        float4 normalWorld = mul(float4(input.normal, 0.0), _World2Object);
        // Final vertex output position
        output.pos = mul(UNITY_MATRIX_MVP, input.vertex);

        // ---------- Local correction ----------------
        output.vertexInWorld = vertexWorld.xyz;
        output.viewDirInWorld = vertexWorld.xyz - _WorldSpaceCameraPos;
        output.normalInWorld = normalWorld.xyz;

        // ---------- Runtime Reflection texture ---------
        output.vertexInScreenCoords = ComputeScreenPos(output.pos);

        return output;
}
```

Local correction ←

Vertex in screen coordinates ←

Rendering Static and Dynamic Reflections - Fragment Shader

```
float4 frag(vertexOutput input) : COLOR
{
    // ----------------- Find reflected vector in World Space ----------------
    float3 DirectionWS = normalize(input.viewDirInWorld); // Interpolated
    float3 normalWS = normalize(input.normalInWorld);
    float3 ReflDirectionWS = reflect(DirectionWS, normalWS);
    // ---------------- Apply local correction to reflection vector ------
    float3 newReflDirectionWS = LocalCorrect(ReflDirectionWS, _BBoxMin, _BBoxMax,
                                             input.vertexInWorld, _EnviCubeMapPos);
    // ---------------- Lookup the color in the static cubemap -------------
    float4 staticReflColor = texCUBE(_Cube, newReflDirectionWS);
    // -------------- Lookup the color in the projected texture produced by the Virtual Reflection Camera -----
    float4 dynReflColor = tex2Dproj( _ReflectionTex, UNITY_PROJ_COORD(input.vertexInScreenCoords) );
    // -------------- Revert blending of reflection camera --------------------
    dynReflColor.rgb /= (dynReflColor.a < 0.00392) ? 1 : dynReflColor.a;
    // -------------- Combine static and dynamic reflections ----------------
    float4 reflCombiColor;
    reflCombiColor.rgb = lerp( staticReflColor.rgb, dynReflColor.rgb, dynReflColor.a );
    reflCombiColor.a = 1.0;

    return reflCombiColor;
}
```

Calculate reflected vector

Apply local correction

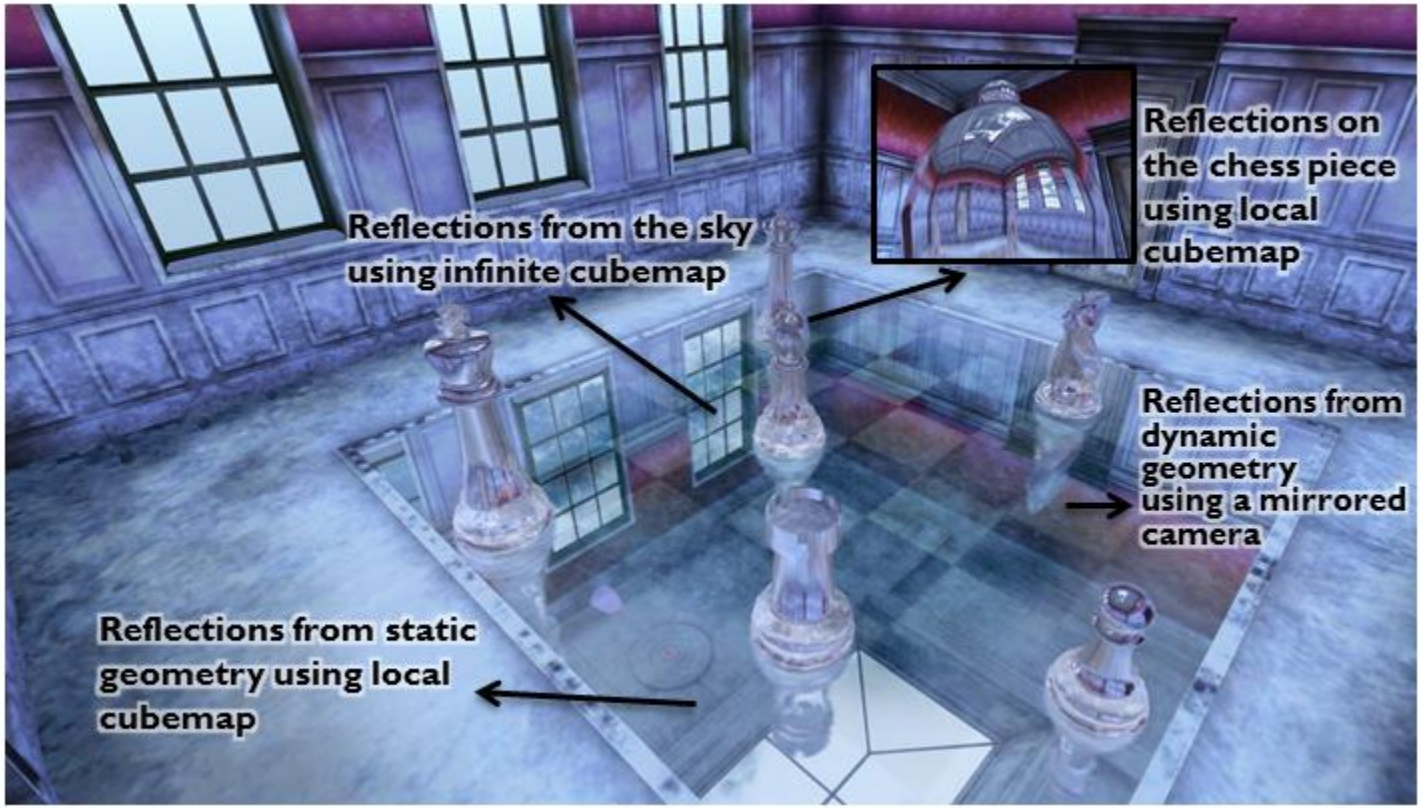
Reflection from static geom.

Reflection from dyn. geom.

Combined reflection

Combined Reflections

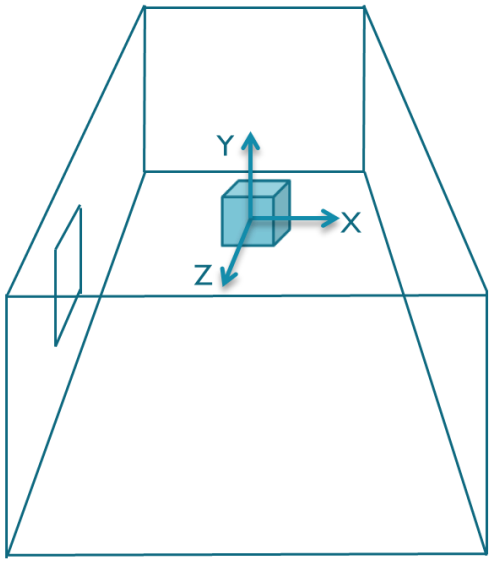
Combined Reflections



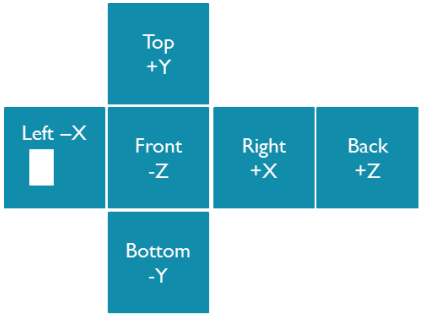
Dynamic Soft Shadows Based on Local Cubemaps

Dynamic Soft Shadows Based on Local Cubemaps

Generation Stage



Render the transparency of the scene in the alpha channel



Camera background alpha colour = 0.

Opaque geometry is rendered with alpha = 1.

Semi-transparent geometry is rendered with alpha different from 1.

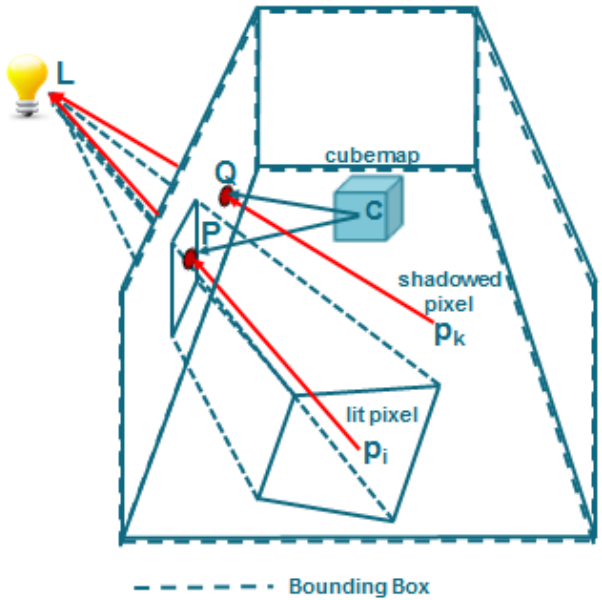
Fully transparent geometry is rendered with alpha 0.

We have a map of the zones where light rays can potentially come from and reach the geometry.

No light information is processed at this stage.

Dynamic Soft Shadows Based on Local Cubemaps

Runtime stage



- Create a vertex to light source L vector in the vertex shader.
- Pass this vector to the fragment shader to obtain the vector from the pixel to the light position $\mathbf{p_iL}$.
- Find the intersection of the vector $\mathbf{p_iL}$ with the bounding box.
- Build the vector **CP** from the cubemap position C to the intersection point P.
- Use the new vector **CP** to fetch the texture from the cubemap.

```
float texShadow = texCUBE(_CubeShadows,  CP).a;
```

Source code in the ARM Guide for Unity Developers at MaliDeveloper.arm.com

Soft Shadows Based on Local Cubemaps - Vertex Shader

```
vertexOutput vert(vertexInput input)
{
        vertexOutput output;

        output.tex = input.texcoord;
        // Transform vertex coordinates from local to world.
        float4 vertexWorld = mul(_Object2World, input.vertex);

        // Transform normal to world coordinates.
        float4 normalWorld = mul(float4(input.normal, 0.0), _World2Object);

        // Final vertex output position.
        output.pos = mul(UNITY_MATRIX_MVP,  input.vertex);

        // ---------- Local correction ----------
        output.vertexInWorld = vertexWorld.xyz;
        output.viewDirInWorld = vertexWorld.xyz - _WorldSpaceCameraPos;
        output.normalInWorld = normalWorld.xyz;
         // ---------- Shadows Static Geometry ----------
        output.vertexToLightInWorld = _LightPosShadows - vertexWorld.xyz;

        return output;
}
```

Passed as varyings to the fragment shader

Soft Shadows Based on Local Cubemaps - Fragment Shader

```
float4 frag(vertexOutput input) : COLOR
{
        float4 texColor = tex2D(_MainTex, float2(input.tex));
        // ---------------- Static Shadows ----------------------------------
        float3 vertexToLightWS = normalize(input.vertexToLightInWorld);
        // Apply the local correction to vertexToLightWS.
        float3 correctVec = LocalCorrect(vertexToLightWS, input.vertexInWorld,
                                         _EnviCubeMapPos, _BBoxMin, _BBoxMax);
        // Fetch the color from the cubemap
        float4 shadowCol = texCUBElod(_Cube,  correctVec);
        // Build a shadow mask to combine with pixel color
        float shadowMask = 1.0 - shadowCol.a;

        float4 finalColor = shadowMask * texColor;
        finalColor.a = 1.0;
        return finalColor;
}
```

Apply local correction

Fetch shadow color

Shadow mask

Combined color

Dynamic Soft Shadows Based on Local Cubemaps

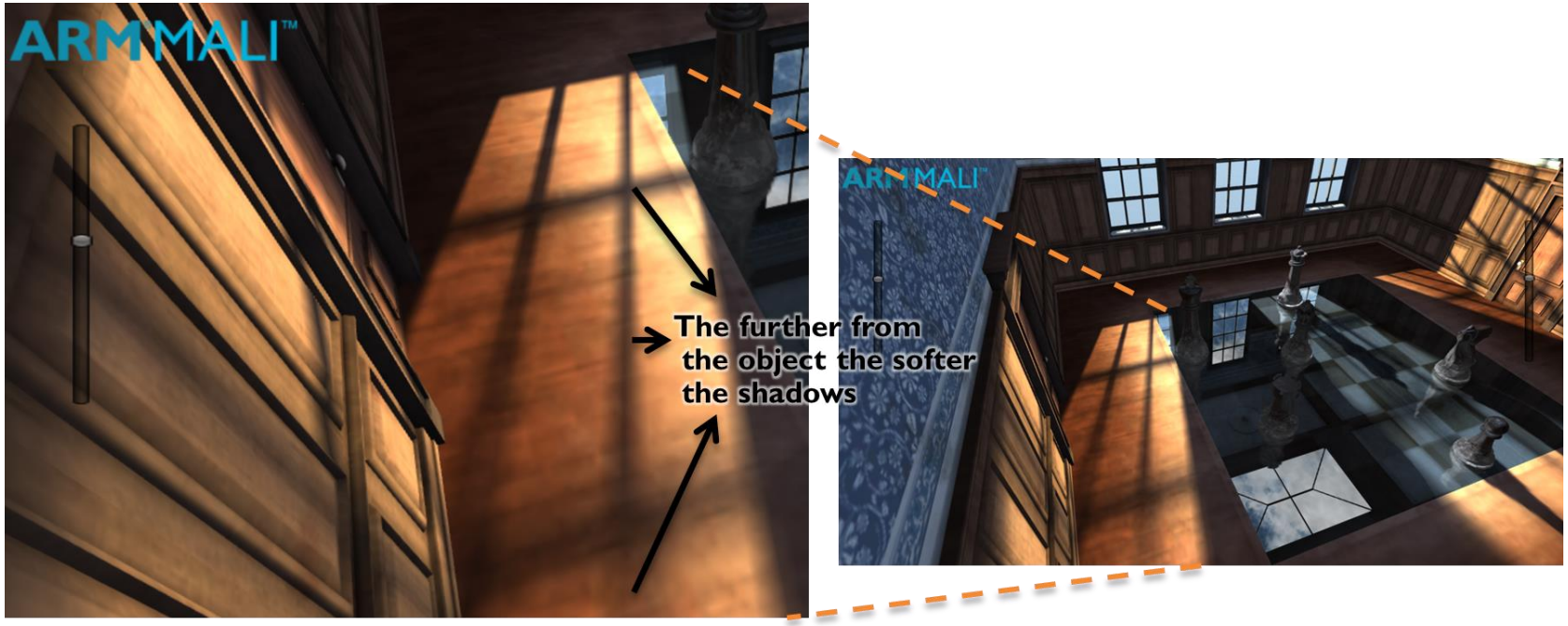
Why dynamic?

If the position of the light source changes, the shadows are generated correctly using the same cubemap.

Dynamic Shadows

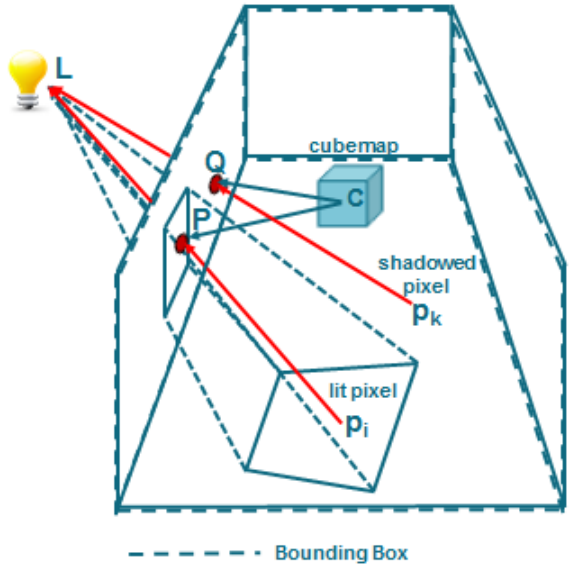
Dynamic Soft Shadows Based on Local Cubemaps

Why Soft?



Dynamic Soft Shadows Based on Local Cubemaps

Why soft?



~~float texShadow = texCUBE(_CubeShadows, CP).a;~~

float4 newVec = float4(CP, factor * length(p_iP))

float texShadow = texCUBElod(_CubeShadows, newVec).a;

Source code in the ARM Guide for Unity Developers at MaliDeveloper.arm.com

ARM

Soft Shadows

Dynamic Soft Shadows Based on Local Cubemaps

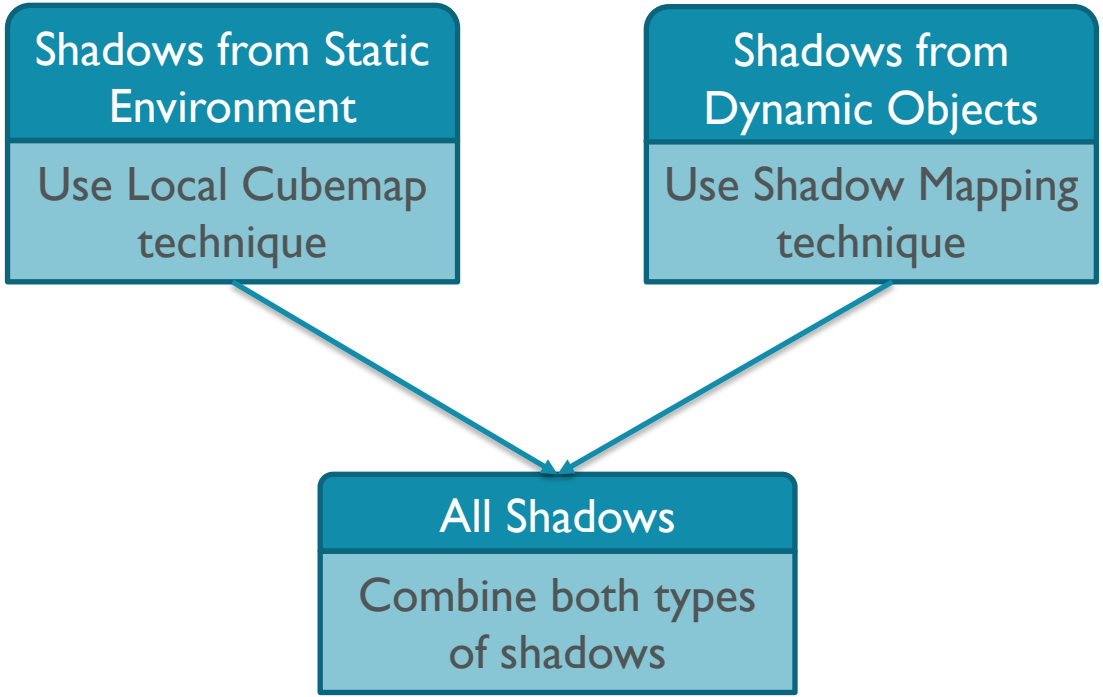
Benefits

1. Simple to implement
2. Very realistic and physically correct
3. High quality of shadows
4. Cubemap texture can be compressed
5. Offline filtering effects can be applied which could be very expensive at run time
6. Resource saving technique compared with runtime generated shadows
7. Very tolerant of deviations from the bounding box shape when compared with reflections

Limitations

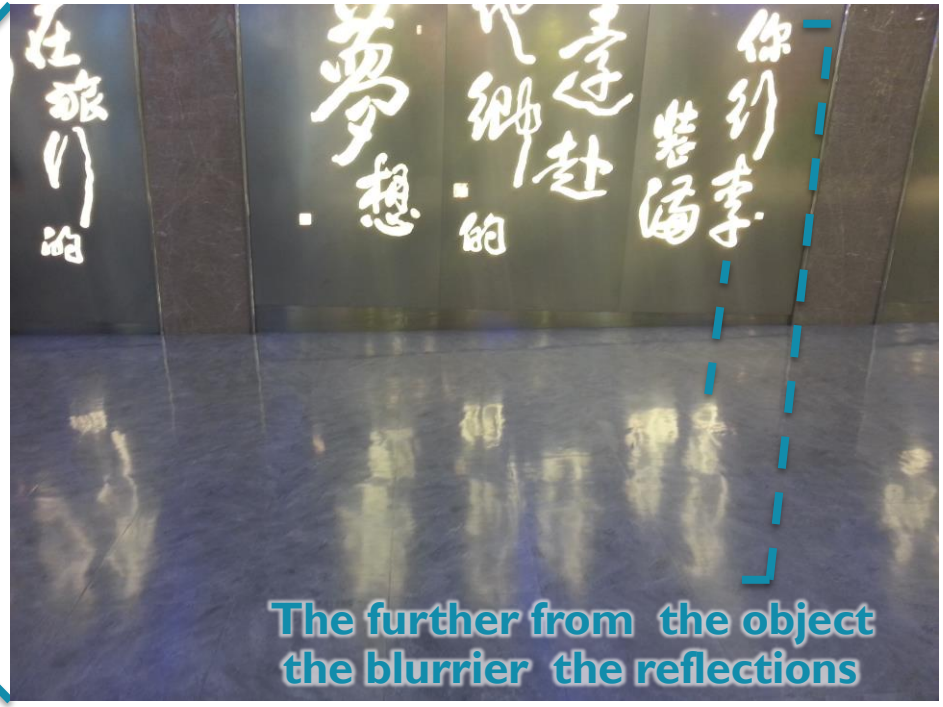
1. Only works well in open plan space with no geometry in the centre where the cubemap will likely be generated
2. For good results, objects in the scene must be close to the proxy geometry when generating static texture
3. Does not reflect changes in dynamic objects unless we can afford to update the cubemap at runtime

Handling Shadows from Different Types of Geometries



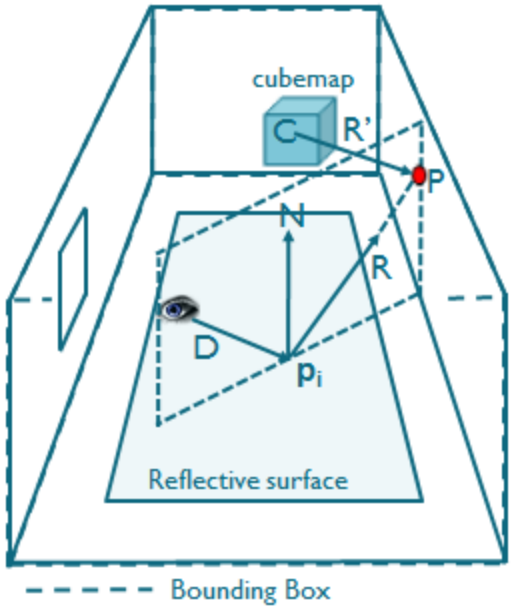
Combined Shadows

Blurred Reflections at the Taiwan Taoyuan International Airport



ARM

Blurred Reflections



float3 R = reflect(D, N);
Find intersection point P
Find vector R' = CP

~~*float4 col = texCUBE(Cubemap, R');*~~

float4 newVec = float4(CP, factor * length(p_iP))

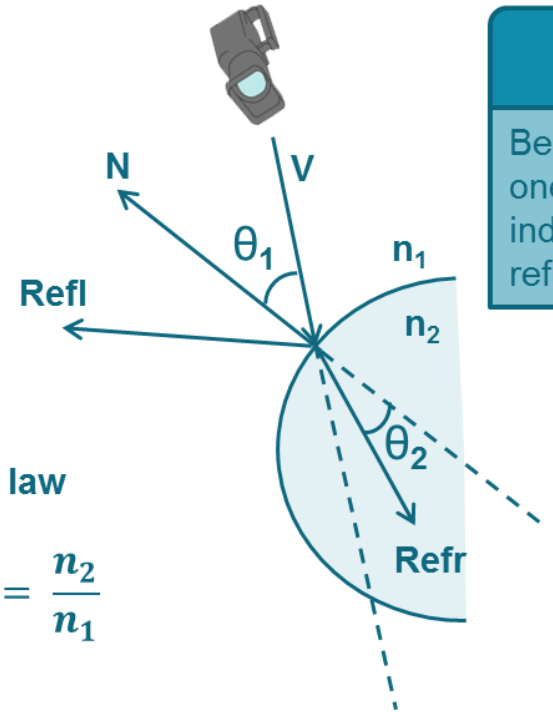
float4 col = texCUBElod(Cubemap, newVec);

ARM

Blurred Reflections Based on Local Cubemaps

Refractions Based on Local Cubemaps

Refraction



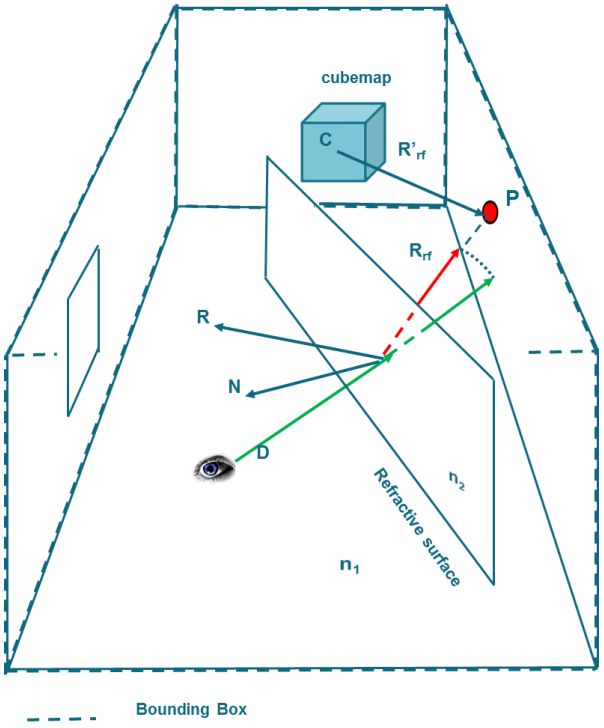
Refraction

Bending of light as it passes from one medium, with refraction index n_1, to another medium with refraction index n_2.

Snell's law

$$\frac{\sin\theta_1}{\sin\theta_2} = \frac{n_2}{n_1}$$

Local Correction to Refraction Vector



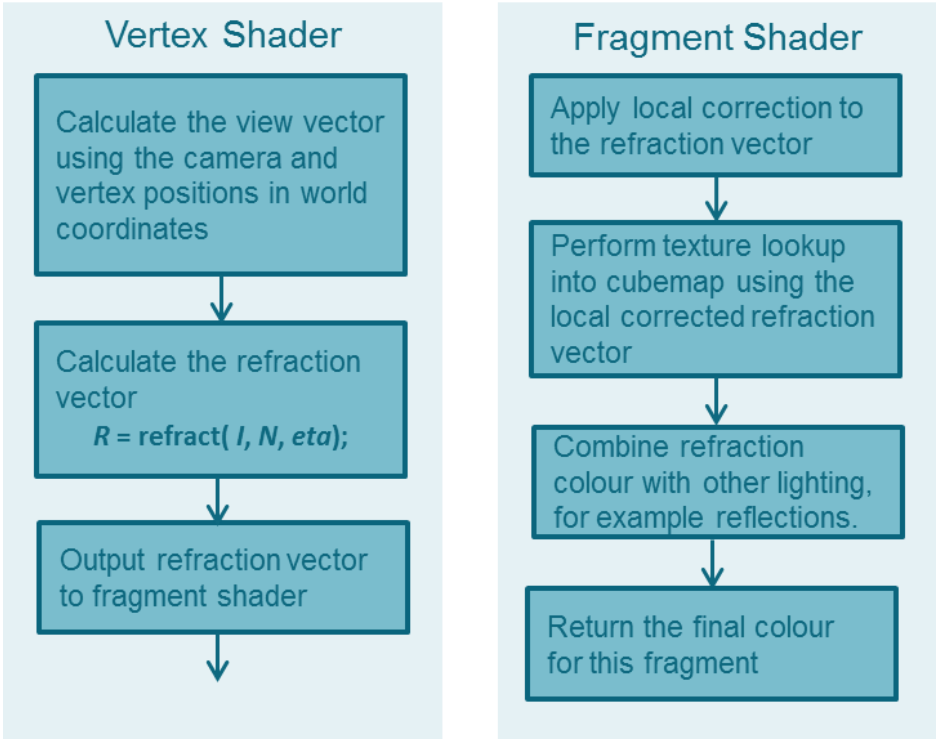
float3 R_{rf} = refract(D_{norm}, N, n_1/n_2);

~~*float4 col = texCUBE(Cubemap, R_{rf});*~~

Find intersection point P
Find vector R'_{rf} = CP
float4 col = texCUBE(Cubemap, R'_{rf});

ARM

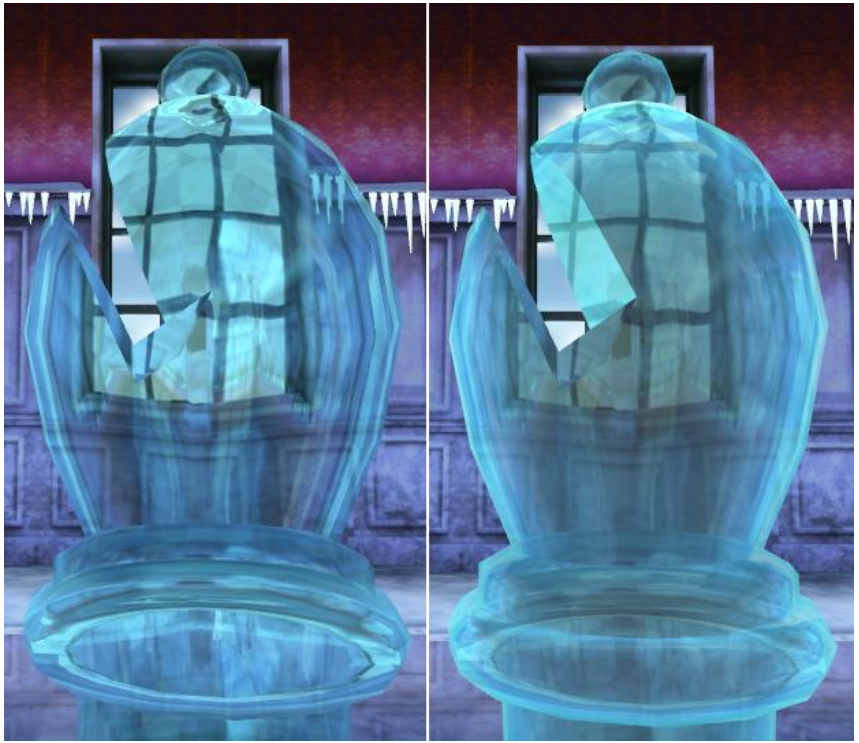
Vertex and Fragment Shaders



Refraction Based on Local Cubemaps combined with reflections

Refraction of semi-transparent object based on local cubemap

- First pass renders transparent object last with front face culling and no depth buffer writing in order to avoid occluding other objects. Object colour is mixed with reflections and background from the local cubemap, calculated with the local correction applied to the refraction vector
- Second pass renders transparent object last with back face culling and Z-write off. Object colour is mixed with reflections and background from the local cubemap calculated with the refracted vector with the local correction applied to refraction vector. Finally alpha blended with the previous pass.



First pass only back faces with local refraction

Second pass only front faces with local refraction and alpha blended with first pass

ARM

Refraction Based on Local Cubemaps in the Ice Cave Demo

Local Cubemap Applications

Reflections based on local cubemaps
2004 - 2012

Dynamic Soft Shadows based on local cubemaps
2014

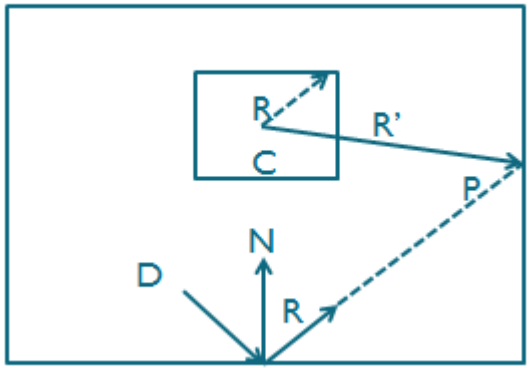
Refraction based on local cubemaps
2015

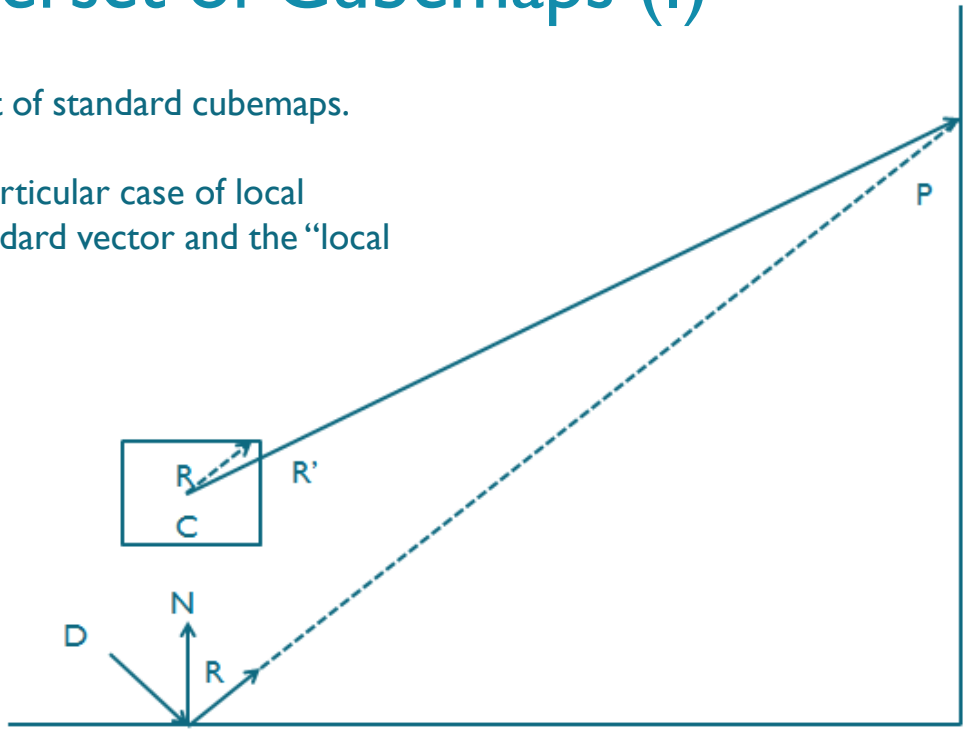
Hopefully more to come …

Local Cubemaps a Superset of Cubemaps (I)

We can think about local cubemaps as a superset of standard cubemaps.

The standard cubemap can be considered as a particular case of local cubemap when the BBox is so large that the standard vector and the "local corrected vector" are the same.





The larger the BBox the more R' approaches to R.

Local Cubemaps a Superset of Cubemaps (II)

$$\boldsymbol{Infinite\ Cubemap} = \lim_{BBoxSize \to \infty} \boldsymbol{Local\ Cubemap}$$

The local correction can be seen as just the correct way of fetching the texture in the LOCAL_CUBEMAP!

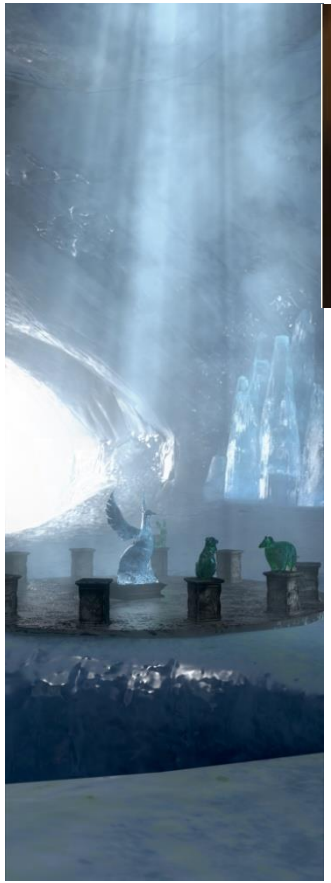
Wrap Up

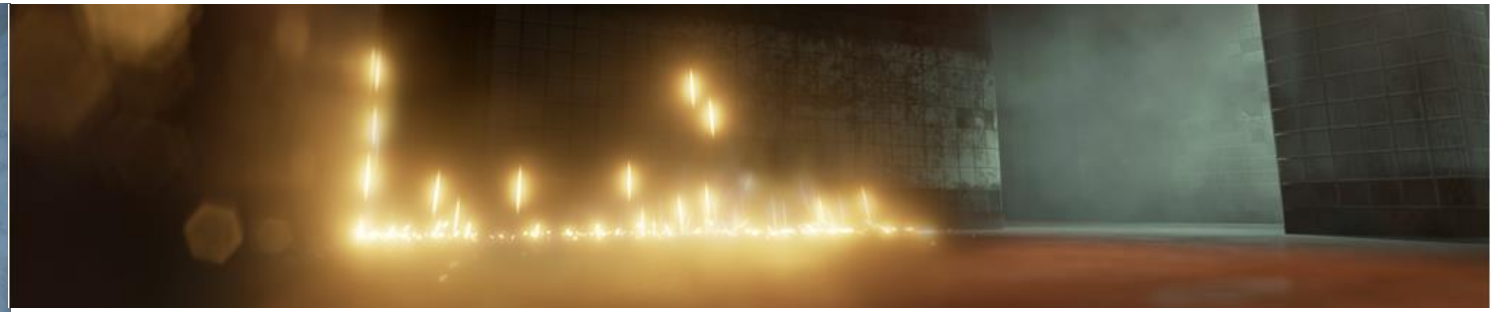
- The concepts of local cubemaps and local correction allow for implementing several highly efficient rendering techniques. These are particularly suitable for mobile devices where runtime resources must be carefully balanced.

- New rendering techniques based on local cubemaps can be effectively combined with other runtime techniques to render different effects for static and dynamic objects together.

- The concept of local cubemap can be considered as a generalization of the standard cubemap. Built-in/hardware support for this feature could improve its performance and usability.

To Find Out More….





- Find out more about techniques based on local cubemaps at:
  - http://malideveloper.arm.com/documentation/developer-guides/arm-guide-unity-enhancing-mobile-games/
  - http://community.arm.com/groups/arm-mali-graphics/blog/2015/04/13/dynamic-soft-shadows-based-on-local-cubemap
  - http://community.arm.com/groups/arm-mali-graphics/blog/2014/08/07/reflections-based-on-local-cubemaps
  - http://community.arm.com/groups/arm-mali-graphics/blog/2015/04/13/refraction-based-on-local-cubemaps
  - http://community.arm.com/groups/arm-mali-graphics/blog/2015/05/21/the-power-of-local-cubemaps-at-unite-apac-and-the-taoyuan-effect

ARM

For more information visit the Mali Developer Centre:

http://malideveloper.arm.com

- Revisit this talk in PDF and audio format post event
- Download tools and resources

Thank you

ARM

Questions