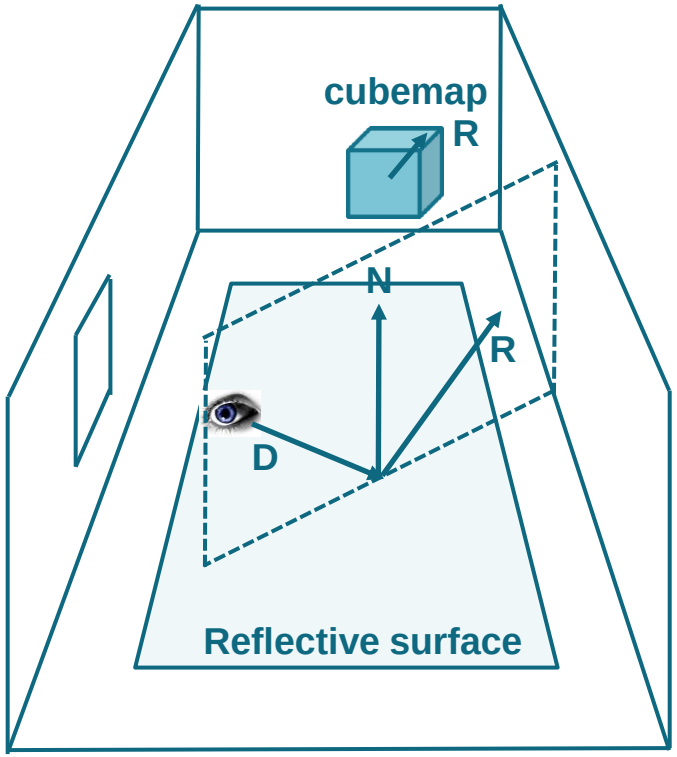
NHTV Workshop
Enhancing your Unity Mobile Games
Shadows Based on Local Cubemaps

Roberto Lopez Mendez
Senior Engineer, ARM

Content

- Local cubemaps
  - Infinite cubemaps. Reflections based on infinite cubemaps.
  - Local correction. Local reflections based on local cubemaps.
- New shadows technique: dynamic soft shadows based on local cubemaps
  - The foundations of shadows based on local cubemaps
  - Why dynamic? Why soft?
  - Benefits and limitations of shadows based on local cubemaps
  - Combining shadows based on local cubemaps with shadows rendered at runtime
- Implementing shadows based on local cubemap in Unity 5
  - Chess room shadows project
  - Shader implementation
  - Build project and deploy on the device
- Wrap up

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);

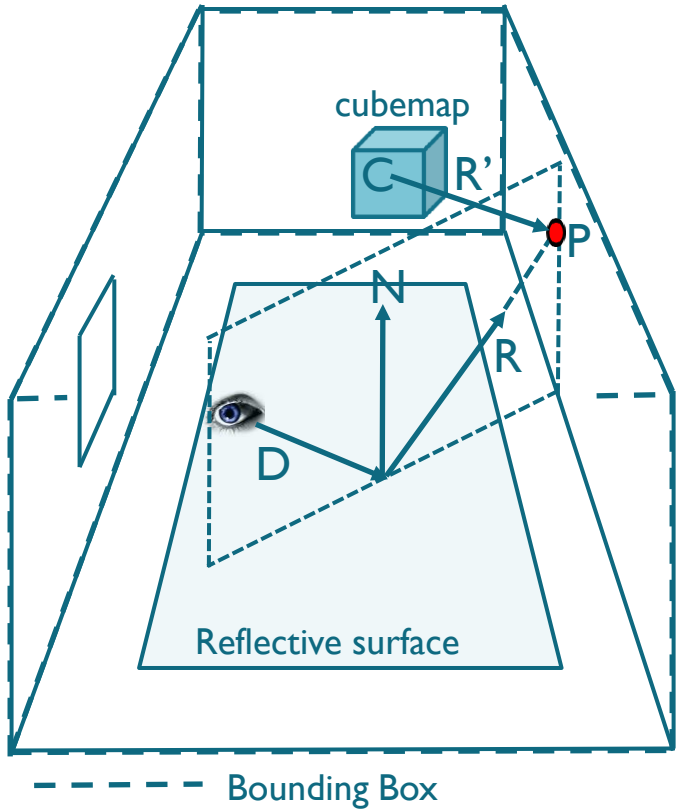
ARM

Incorrect Reflections

Reflection generated using a cubemap without any local binding

ARM

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');

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/

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

ARM

Correct Reflections

Reflection generated after applying the "*local correction*"

Reflection generated without "*local correction*"

ARM

Infinite and Local Cubemaps

Infinite Cubemaps

- They are used to represent the lighting from a distant environment.
- Cubemap position is not relevant.

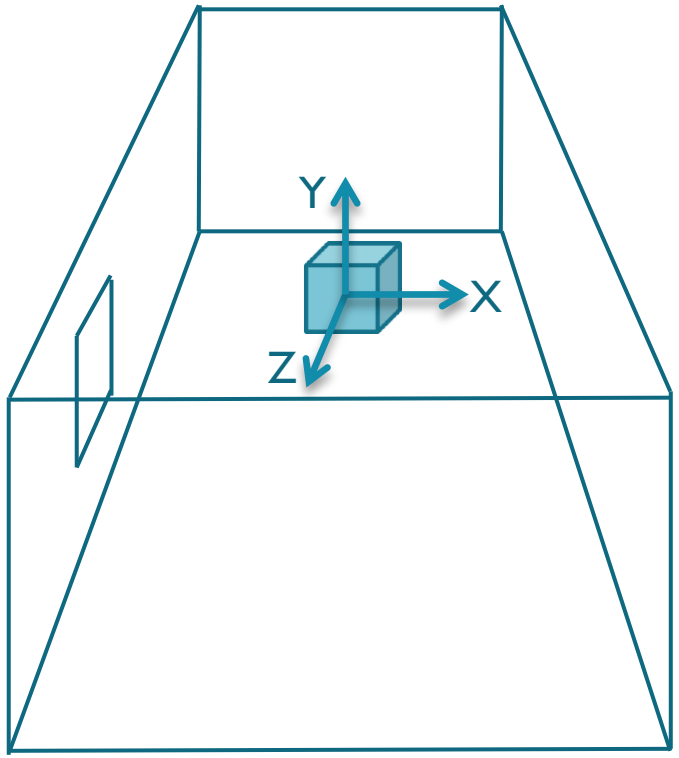
Local Cubemaps

- They are used to represent the lighting from a finite local environment.
- Cubemap position is relevant.
- The lighting from these cubemaps is right only at the location where the cubemap was created.
- *Local correction* must be applied to get the right local reflections.

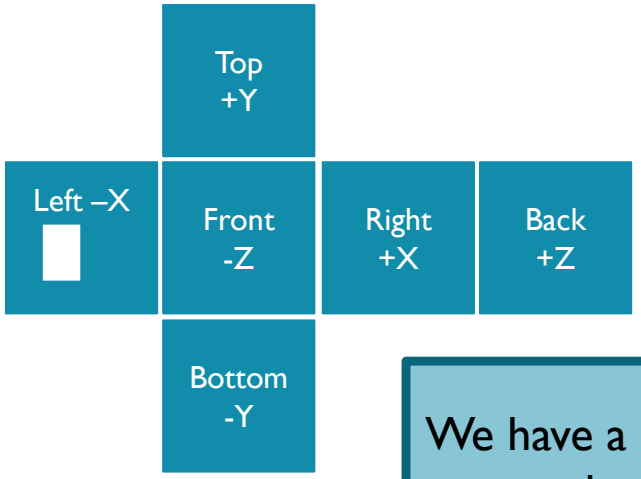
Dynamic Soft Shadows

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.

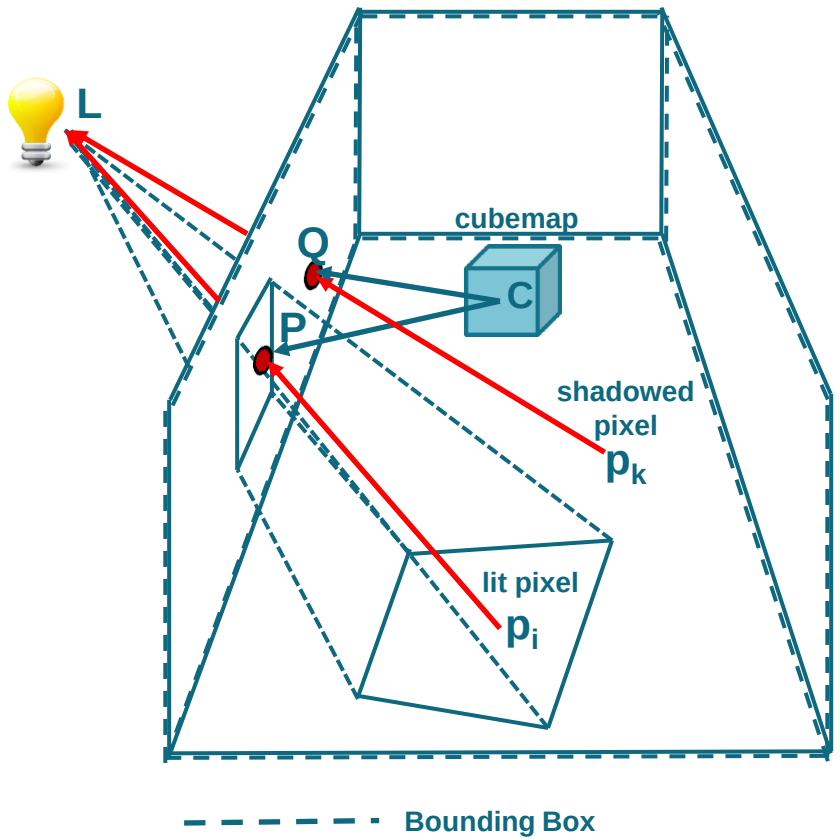
We have a map of the zones where light rays can potentially come from and reach the geometry.

No light information is processed at this stage.

ARM

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.ar.m.com

ARM

Simple 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);

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

        // ---------- Local correction ------------
        output.vertexInWorld = vertexWorld.xyz;
        output.vertexToLight = _ShadowsLightPos - output.vertexInWorld;

        return output;
}
```

Passed as varyings to the fragment shader

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

ARM

Simple Fragment Shader

```
float4 frag(vertexOutput input) : COLOR
{
        float4 texColor = tex2D(_MainTex, input.tex.xy);

        // Normalize vertex to light vector in WS.
        float3 normVertexToLight = normalize(input.vertexToLight);

        // Working in World Coordinate System.
        float3 localPosWS = input.vertexInWorld;
        float3 intersectMaxPointPlanes = (_BBoxMax - localPosWS) / normVertexToLight ;
        float3 intersectMinPointPlanes = (_BBoxMin - localPosWS) / normVertexToLight ;
        // 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 + normVertexToLight * distToIntersect;

        // Get local corrected vertex-to-light vector.
        float3 localCorrectVertexToLight = intersectPositionWS - _EnviCubeMapPos;

        // Lookup the cubemap texture with the correct vector.
        float shadowVal = texCUBE(_Cube, localCorrectVertexToLight ).a;
        shadowVal = 1.0 - shadowVal ;
        float4 shadowColor = float4(shadowVal , shadowVal , shadowVal , 1.0);
        // Combine colours to get the final pixel colour with other colours.
        return _AmbientColor * texColor + texColor * shadowColor;
}
```

Normalize vertex-to- light vector

Ray-box intersection algorithm

Local corrected reflected vector

Fetch texture value at the alpha channel

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

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.

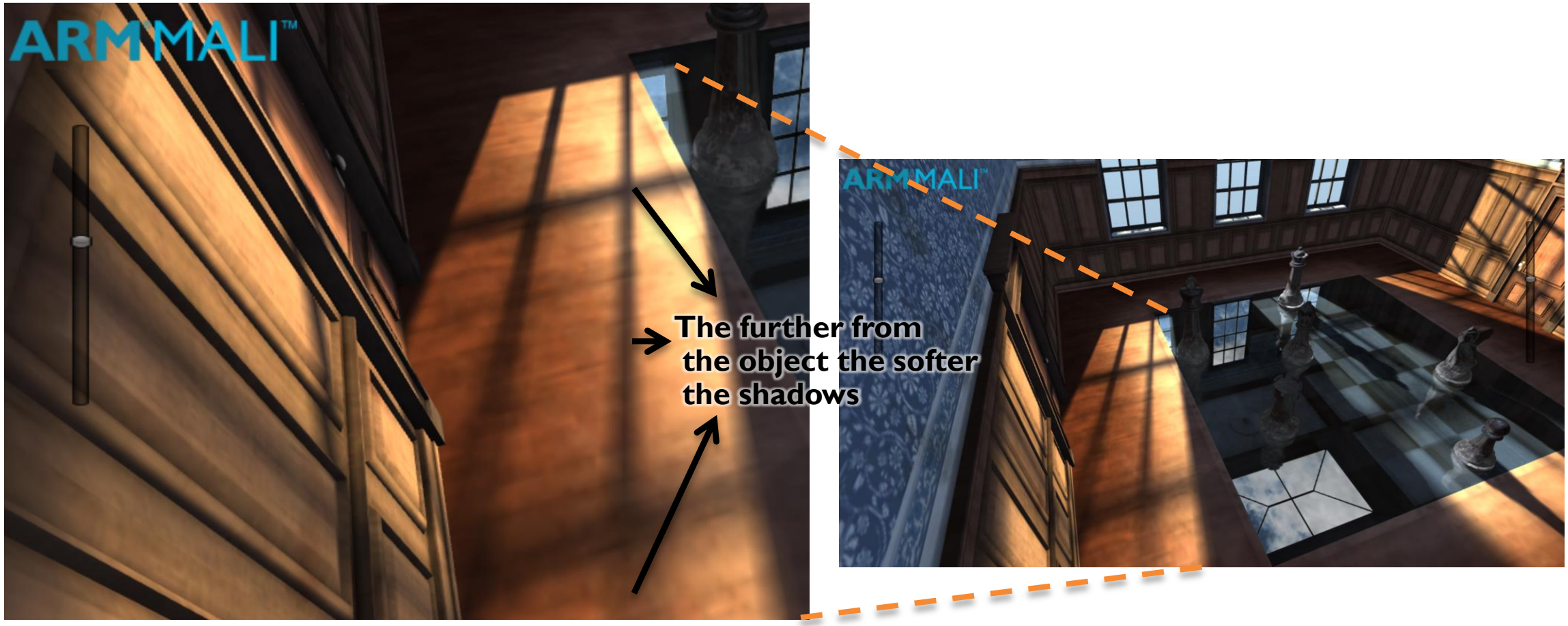
ARM

Dynamic Shadows

ARM

Dynamic Soft Shadows Based on Local Cubemaps

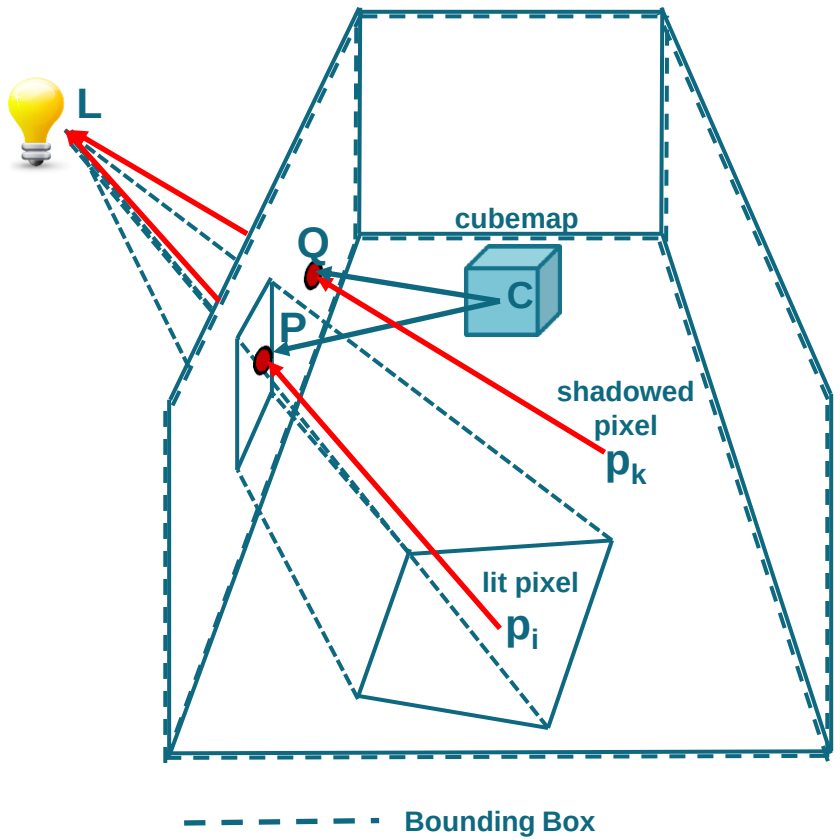
Why soft?



ARM

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.ar.m.com

ARM

Soft shadows

ARM

Dynamic Soft Shadows Based on Local Cubemaps

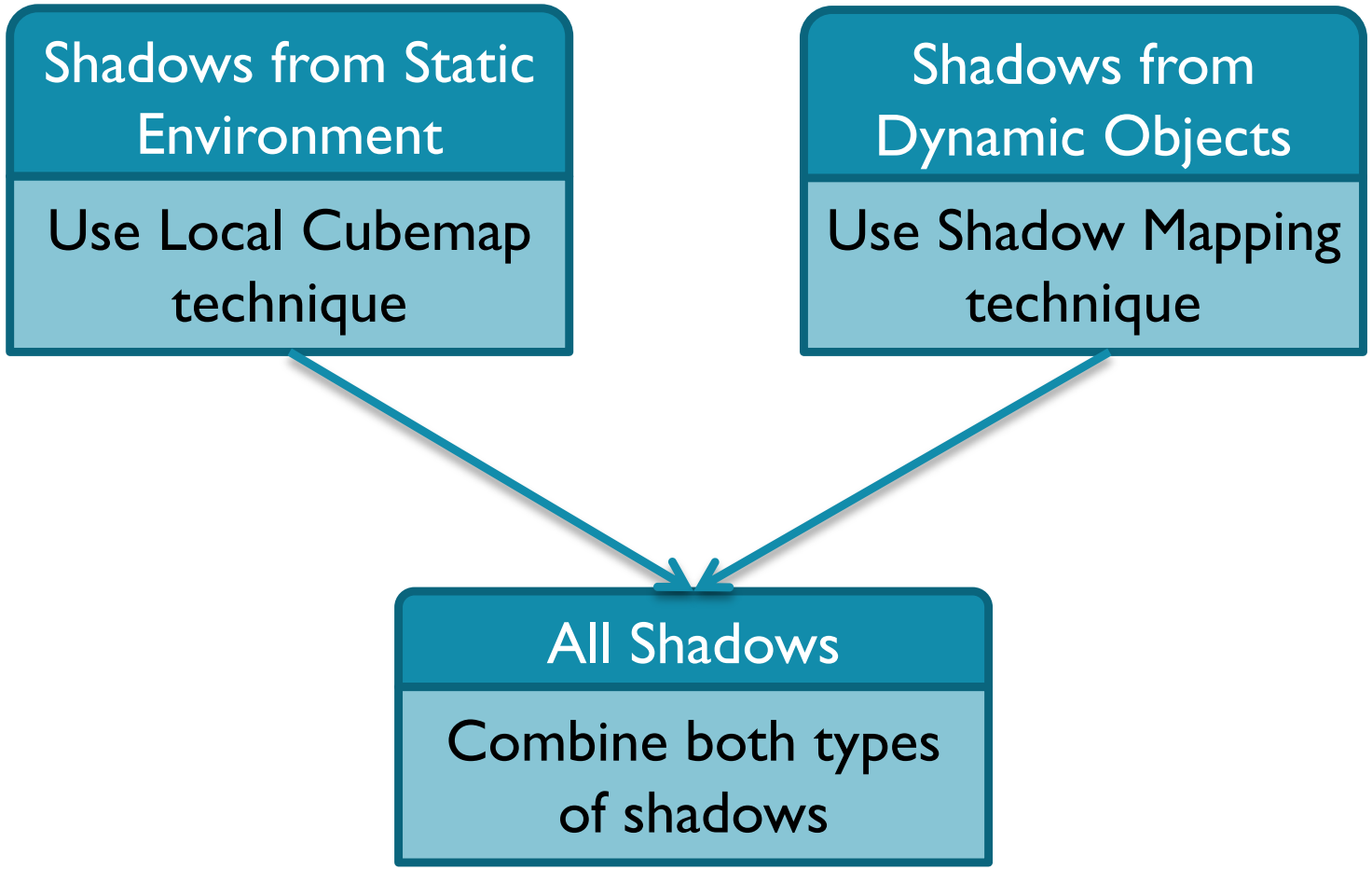
Benefits

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. Works fine in open space with no geometry in the centre from where the cubemap will more likely be generated.
2. Objects in the scene must be close to the proxy geometry when generating static texture for good results.
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



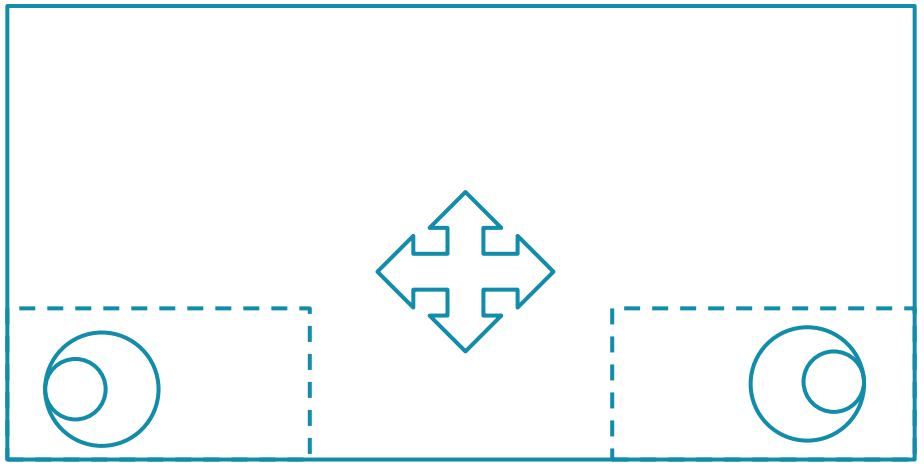
ARM

Combined shadows

ARM

Unity 5project elements

- Chess room project
  - Static geometry: chess room
  - Dynamic geometry: chess piece
  - Main camera
  - Shadow camera
  - Shaders
  - UI elements: joysticks and sliders
  - Touch events to move the chess piece



Wrap Up

- The new shadows rendering technique based on local cubemaps is a resource saving techniques that work great in mobile devices where available resources must be carefully balanced.

- This techniques can be effectively combined with runtime techniques to render static and dynamic objects together.

- Implementing this technique in Unity 5 is very simple.

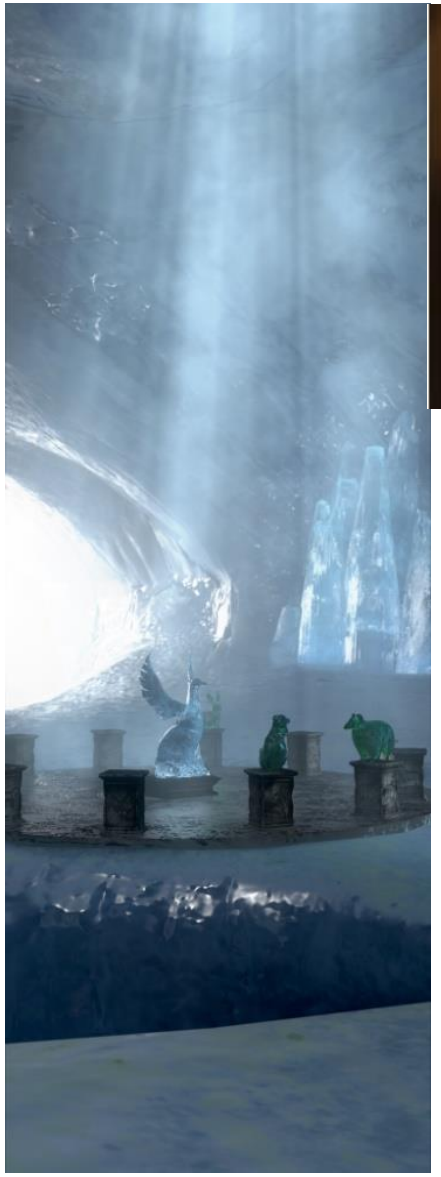
More source code in the update of ARM Guide for Unity Developers at Unite Boston 2015.

ARM

Demo Ice Cave

ARM

To Find Out More….



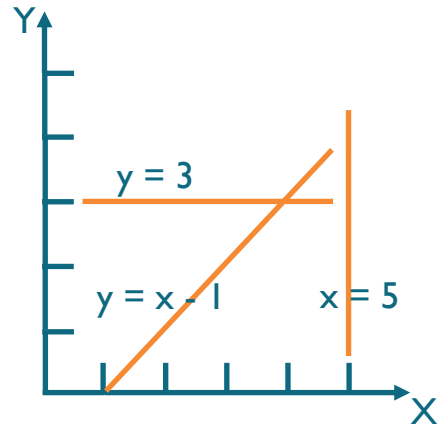
- Find out more about shadows based on local cubemaps at:
  - http://community.arm.com/groups/arm-mali-graphics/blog/2015/04/13/dynamic-soft-shadows-based-on-local-cubemap
- Find out more about other techniques based on local cubemaps at:
  - http://community.arm.com/groups/arm-mali-graphics/blog/2014/08/07/reflections-based-on-local-cubemaps
  - 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

Thank You

Ray-Box Intersection (I)

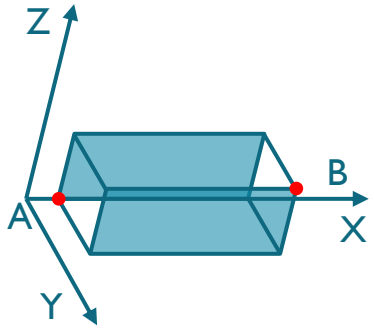
Equation of a line
$y = mx + b$



Vector form:
$r = O + t*D$

O – origin point
D – direction vector
t – parameter

An axis aligned bounding box AABB
can be defined by its min and max points (A, B)



The AABB defines a set of lines parallel to coordinate axis. Each component of line can be defined by the equation:
$x = A_x;\ y = A_y;\ z = A_z$
$x = B_x;\ y = B_y;\ z = B_z$

To find where a ray intersects one of those lines we just equal both equations:
$O_x + t_x*D_x = A_x$ for example.
The solution can be written as:
$t_{Ax} = (A_x - O_x) / D_x$

In the same way we can obtain the solution for all components of both intersection points:

$t_{Ax} = (A_x - O_x) / D_x$
$t_{Ay} = (A_y - O_y) / D_y$
$t_{Az} = (A_z - O_z) / D_z$

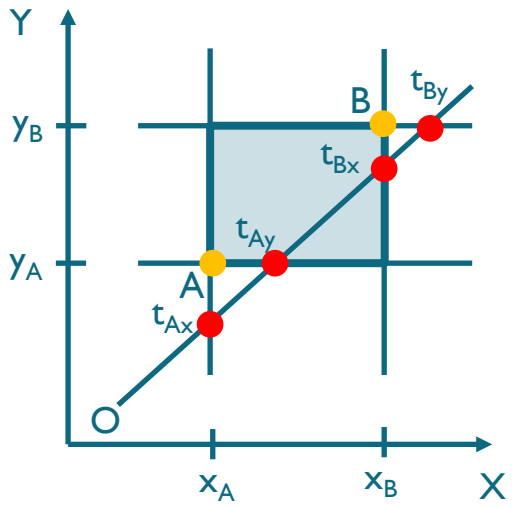
$t_{Bx} = (B_x - O_x) / D_x$
$t_{By} = (B_y - O_y) / D_y$
$t_{Bz} = (B_z - O_z) / D_z$

In vector form:
$t_A = (A - O) / D$
$t_B = (B - O) / D$

We have found in this way where the line intersects the planes defined by the faces of the cube but it doesn't means that the intersections lie on the cube.

Ray-Box Intersection (II)

2D Representation



We need to find which of the solutions is really an intersection with the box.

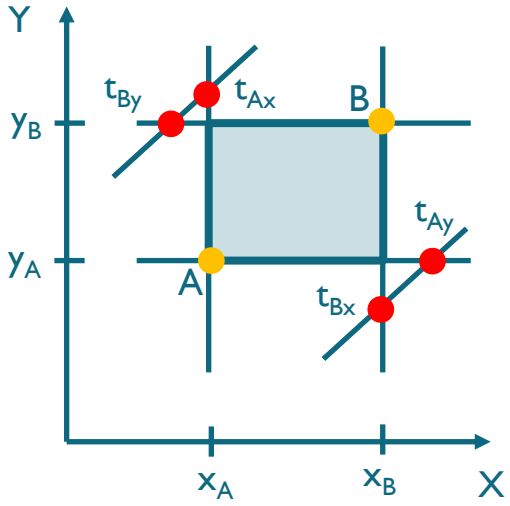
We need the greater value of t parameter for the intersection at the min plane.

$t_{min} = (t_{Ax} > t_{Ay})\ ?\ t_{Ax} : t_{Ay}$

We need the smaller value of t parameter for the intersection at the max plane.

$t_{min} = (t_{Ax} > t_{Ay})\ ?\ t_{Ax} : t_{Ay}$

We also must consider those cases when we get no intersections.



Detailed explanation of handling all the cases in 3D can be found elsewhere .

Nevertheless if we guarantee that the origin of our vector R is enclosed by the BBox (i.e the origin of the R ray is inside the BBox) then we always have two intersections with the box and the handling of different cases is simplified.