# Implementing Reflections in Unity Using Local Cubemaps

Roberto Lopez
Senior Engineer

# Content

- Environment Mapping.
- Cubemaps.
- Infinite and Local Cubemaps.
- Reflections with Infinite Cubemaps.
- Local Correction.
- Implementation of Local Correction in the Shader.
- Unity Project ReflLab
- Running ReflLab in Nexus-10
- Filtering cubemaps
- Conclusions

# Environment Mapping

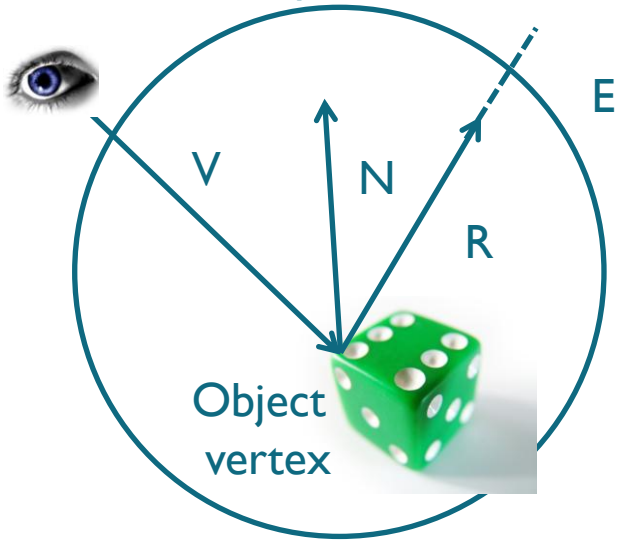
Environment mapping simulates reflections or lighting upon objects without going through expensive ray-tracing or lighting calculations.

## Spherical Environment mapping

A clever way of doing reflections in real time.

Limitations:
Distortions when mapping a picture onto a sphere.

Environment map on a sphere




The spherical surface is mapped into 2D:

$$u = \frac{R_x}{m} +$$

$$v = \frac{R_y}{m} +$$

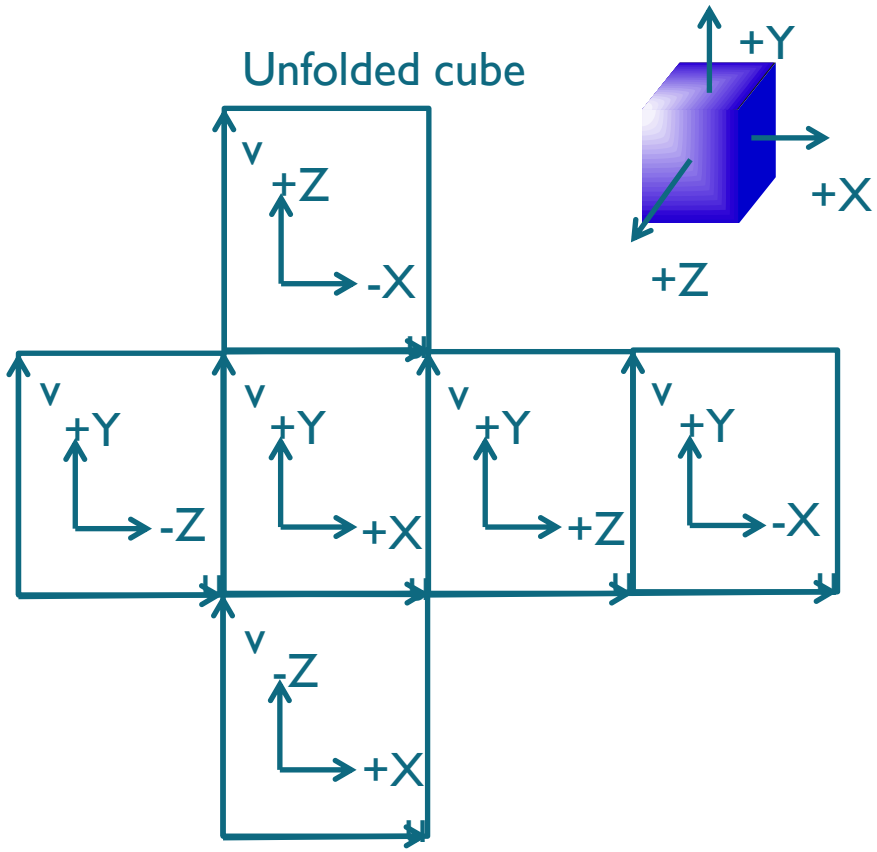
$$m = 2.\sqrt{R_x^2 + R_y^2 + (R_z$$

# Why Cubemaps?

In 1999 was possible to use cubemaps with hardware acceleration.

## Cubemaps

- Hardware accelerated.
- They solved the problems of image distortions, viewpoint dependency and computational inefficiency.




Source images are sampled directly. No distortion introduced by resampling into an intermediate environment map.

# Infinite and Local Cubemaps

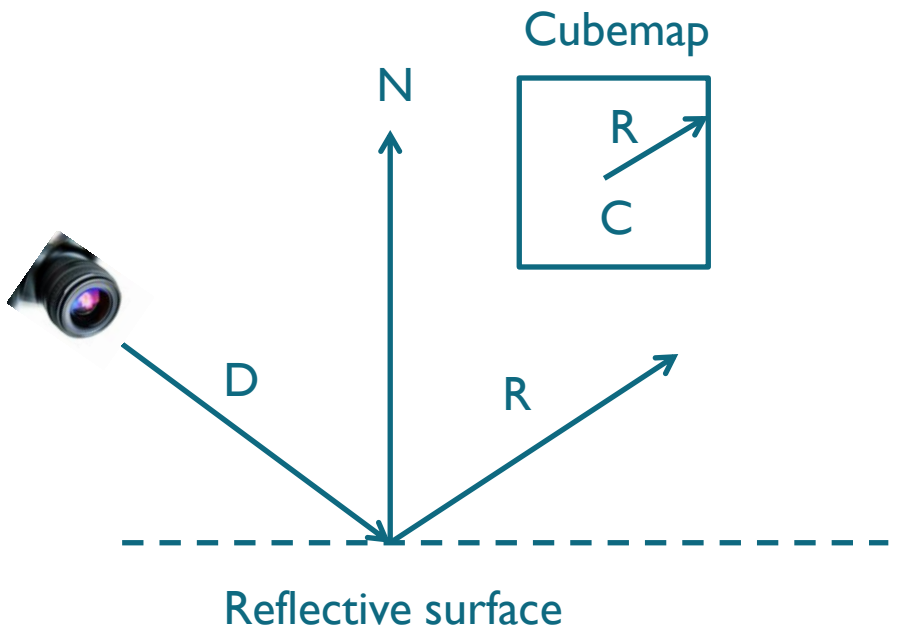
## Infinite Cubemaps

- They are used to represent the lighting from a distant environment.
- Cubemap position is not relevant.
- They are used mainly for outdoor lighting. For example skyboxes.

## 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 adapt the intrinsic infinite nature of cubemaps to local environment.

# Reflections with Infinite Cubemaps




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

In the fragment shader we fetch the texture colour from the cubemap:

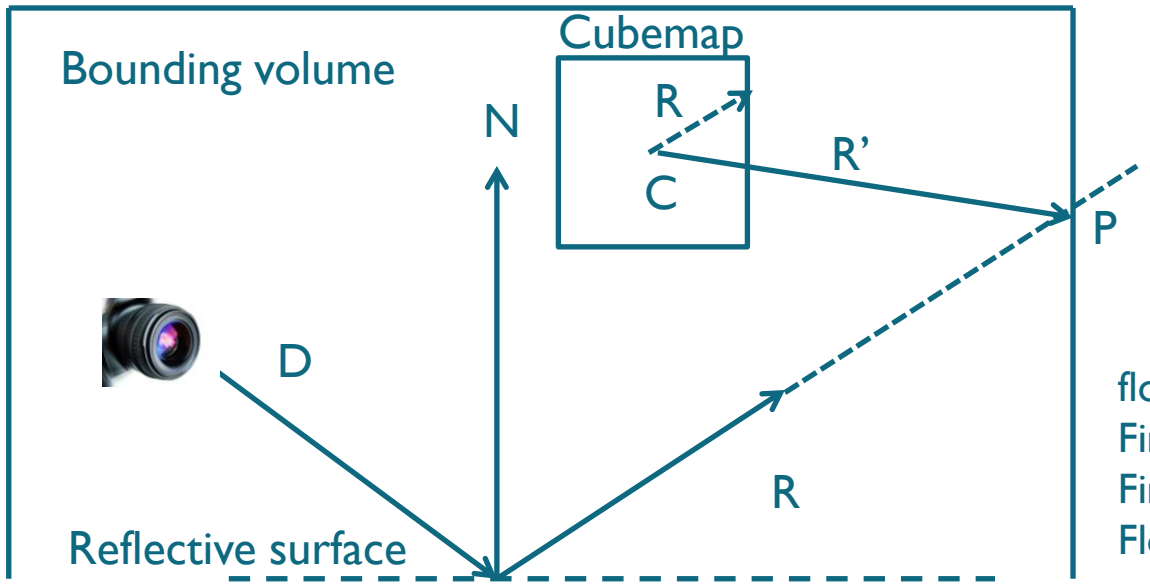
```
float3 R = reflect(D, N);
float4 col = texCube(Cubemap, R);
```

# Wrong Reflections

Reflections generated using a cubemap without any local binding.

# Local Correction

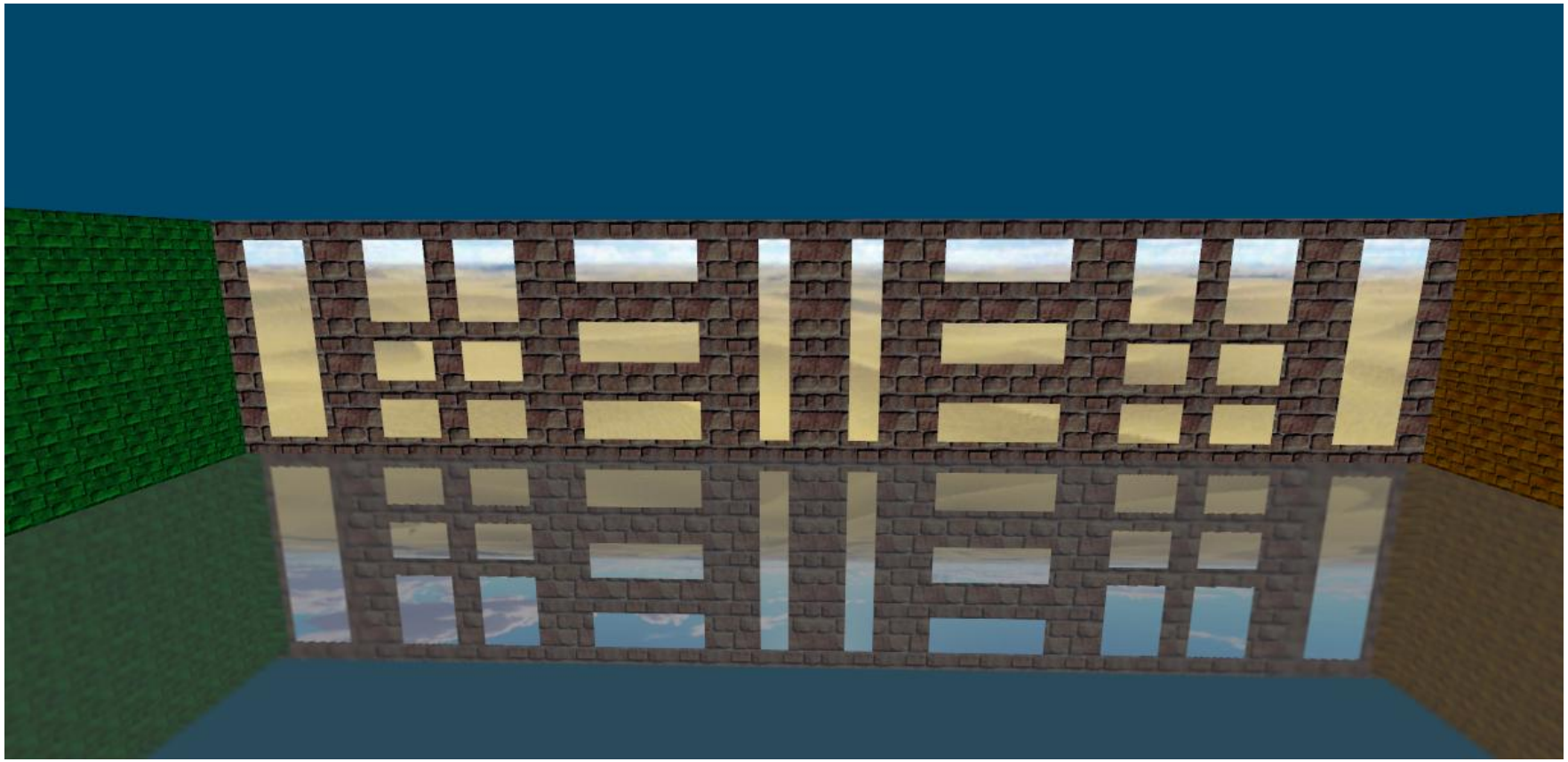
Instead of fetching the texel from the cubemap using the vector R we find where the vector intersects the bounding box and build a new vector from the centre of the cubemap to the intersection point and use this new vector to fetch the texture colour from the cubemap.




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

# Right Reflections



Reflections generated  after applying the "*local correction*".

# Reflections based on Local Cubemaps

## Advantages

1. Simple to implement.
2. Very realistic.
3. Physically correct.
4. High quality of reflections. No pixel flickering.
5. Resource saving technique. Important for mobile devices.
6. Cubemap - Bake once reuse many times.
7. Additional filtering effects can be added which is very expensive at run time.

## Limitations

Valid only to simulate reflections of static geometry.

Solution: Generate reflections of dynamic objects at run time and combine with reflections based on static 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;

        return output;
}
```

# Fragment Shader

```
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;
}
```

# Loading Project in Unity

## Loading Project

1. Launch Unity.
2. File => Open Project => Open Other, navigate to the desktop, find …/Desktop/UnityProjects/ReflectionWorkshop/ folder folder and select it.
3. In Unity project explorer double click on Assets/ReflLab to load the scene.

## Loading Solution

1. Launch MonoDevelop Assets => Sync MonoDevelop Project.
2. If needed load manually the …/Desktop/UnityProjects/ReflectionWorkshop/ /ReflectionWorkshop.sln.
3. Select View => Visual Design.
4. Explore solution files in the left panel.

# Deploying the Application to the Device

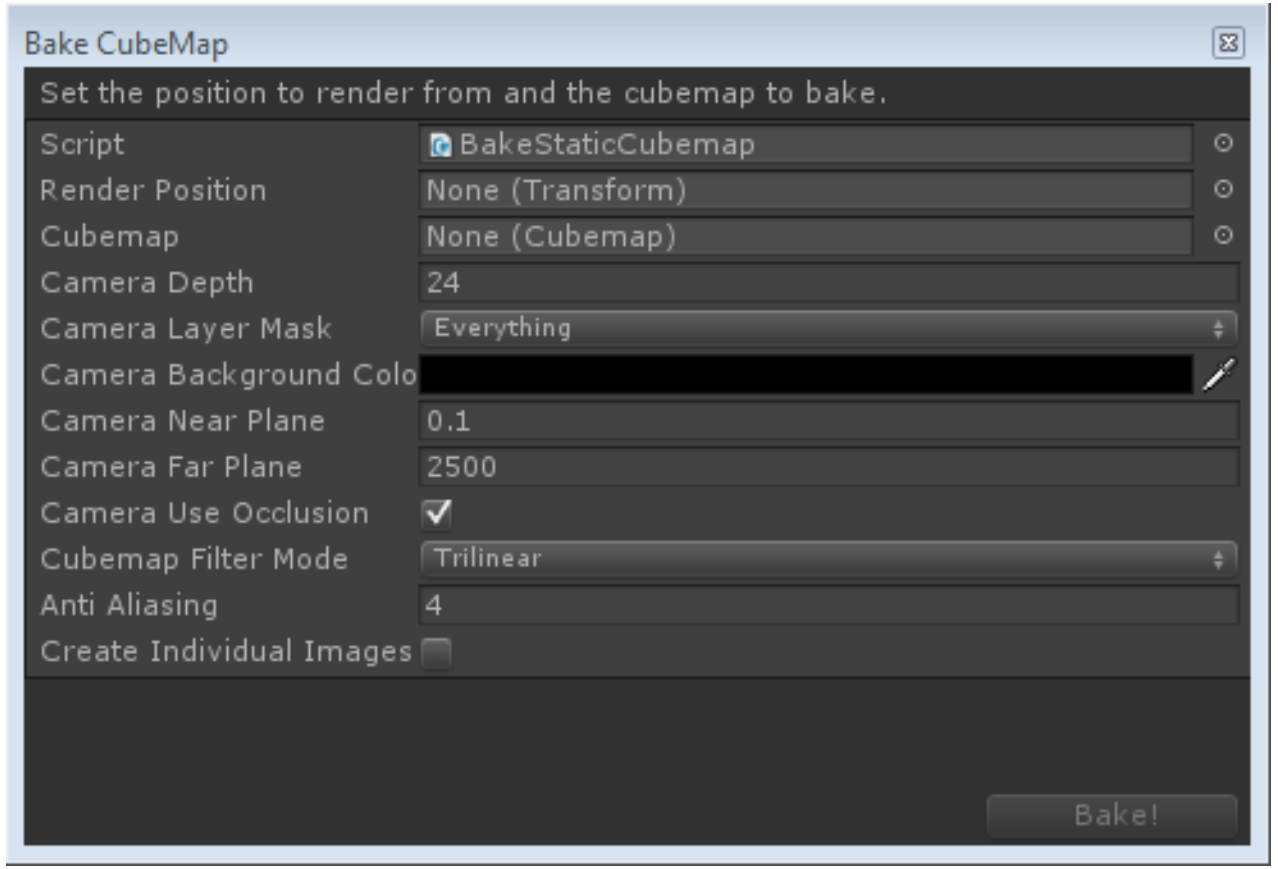
**Building and Deploying**

1. Connect the device to the laptop with the USB cable.
2. Wait until the device is detected.
3. Launch command prompt and execute "adb devices" command to check if the device has been recognized.
4. In Unity press Ctrl + C to build and deploy the application to the Android device.

# Generating Cubemaps

## Bake Cubemap Editor Tool

- Available from GameObject menu.
- Create the cubemap.
- Launch Bake Cubemap Tool.
- Provide camera render position and the cubemap.
- Optionally check the box to save individual images if filtering will be applied to cubemap.




Screenshot from the Unity Editor tool to bake cubemaps.

# Filtering Cubemaps

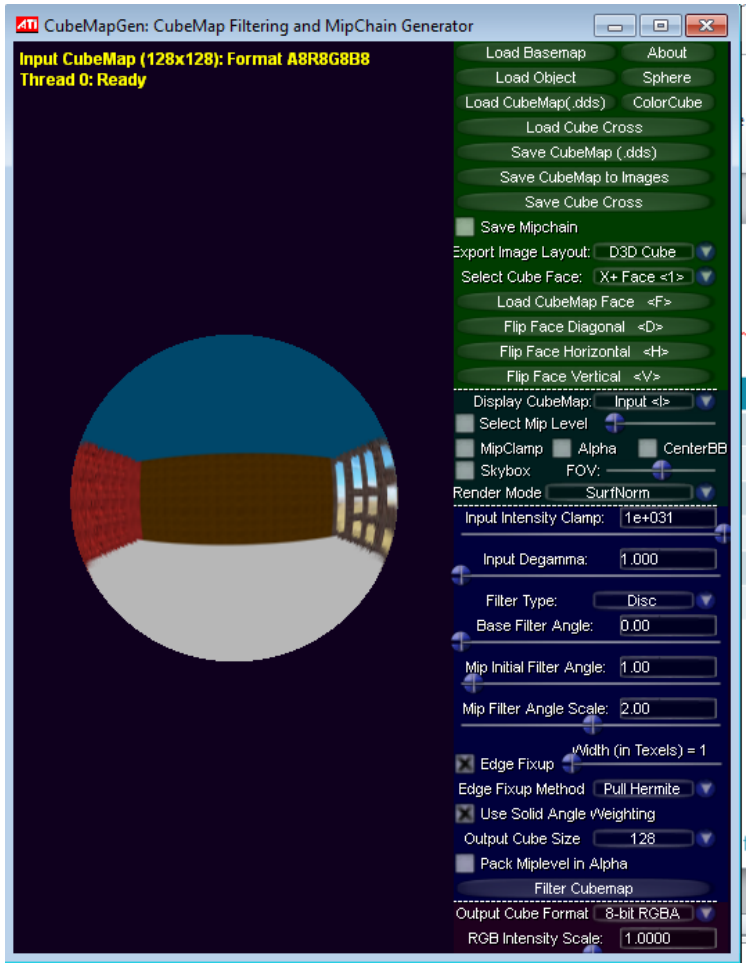
## Unity to AMD CubeMapGen

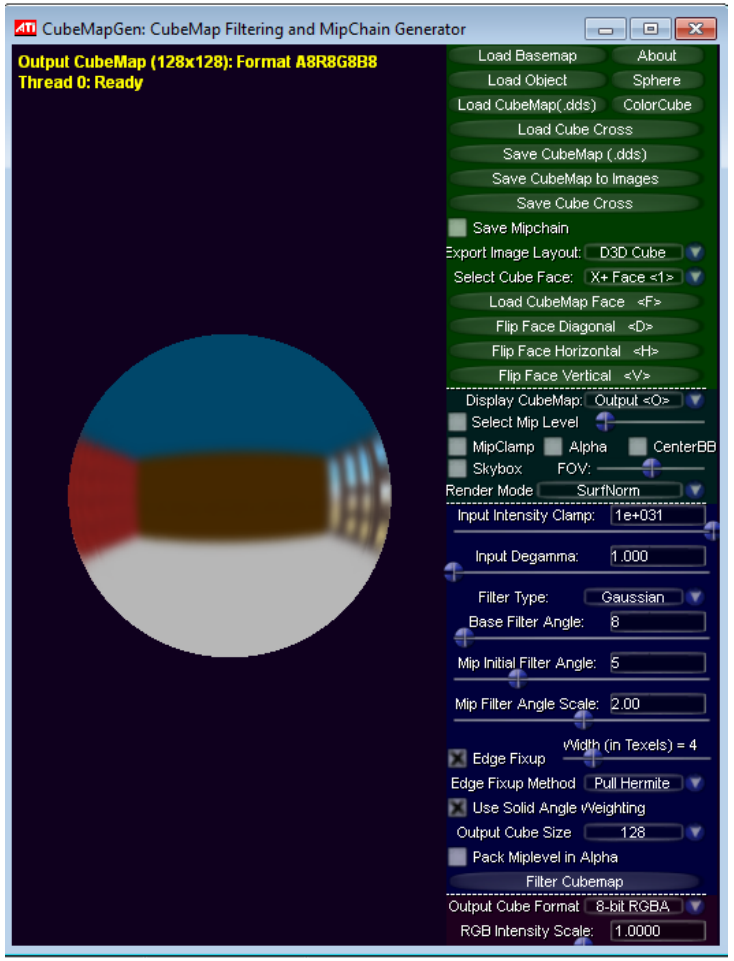
- Check the box to save individual images when baking the cubemap.
- Launch CubeMapGen tool and load cubemap images following the relations given in the table.
- Save cubemap as a single dds or cube cross image for easy loading if needed when experimenting with filters as undo is not available.
- Apply any filter to cubemap.
- Save cubemap as individual images.

| AMD CubeMapGen | Unity |
|---|---|
| X+ | -X |
| X- | +X |
| Y+ | +Y |
| Y- | -Y |
| Z+ | +Z |
| Z- | -Z |

| Filter Settings | Value |
|---|---|
| Type | Gaussian |
| Base Filter Angle | 8 |
| Mip Initial Filter Angle | 5 |
| Mip Filter Angle Scale | 2.0 |
| Edge Fixup | checked |
| Edge Fixup Width | 4 |

# Filtering Cubemaps




Screenshots from AMD CubeMapGen tool.

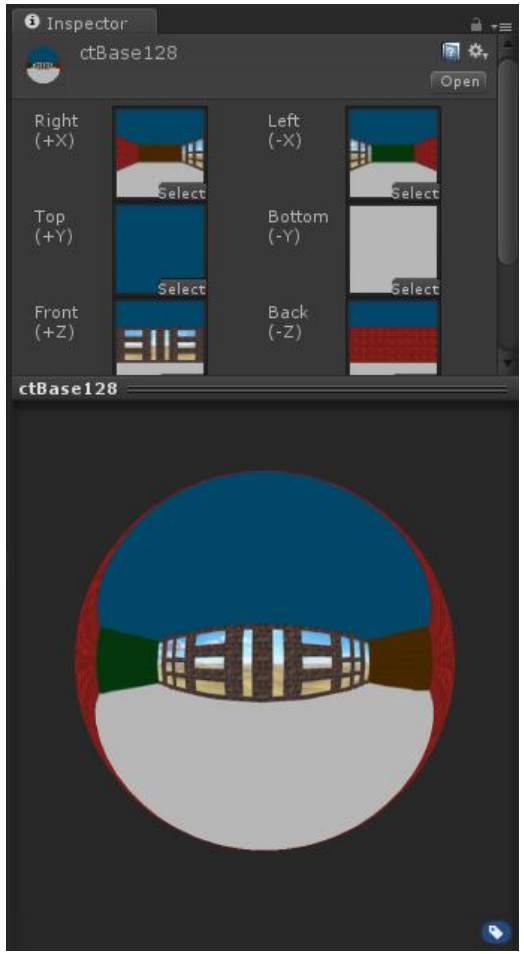
# Filtering Cubemaps

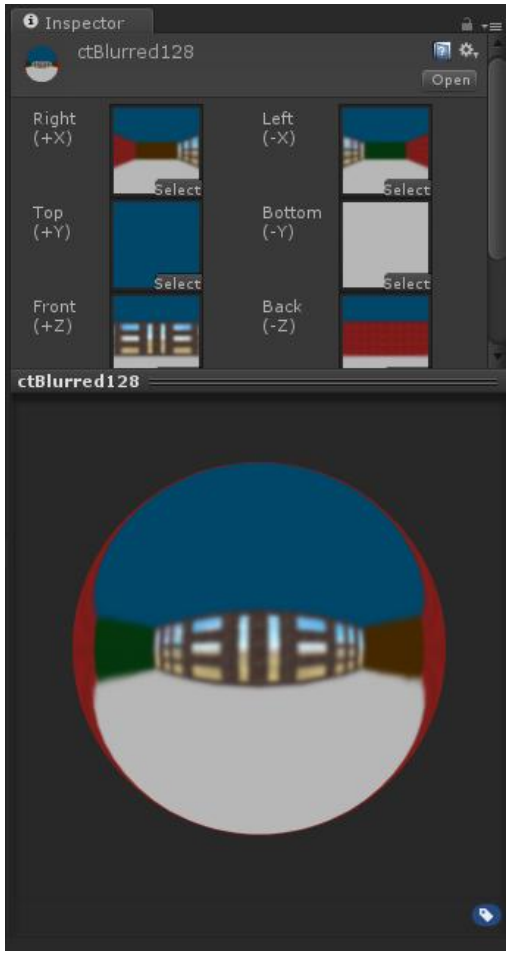
## AMD CubeMapGen to Unity

- Flip vertically all filtered images with any image processing tool.
- Import filtered images in Unity.
- Create new empty cubemap in Unity.
- Populate cubemap images according with the table.

| AMD CubeMapGen | Unity |
|---|---|
| .c00 | +X |
| .c01 | -X |
| .c02 | +Y |
| .c03 | -Y |
| .c04 | +Z |
| .c05 | -Z |

# Filtering Cubemaps

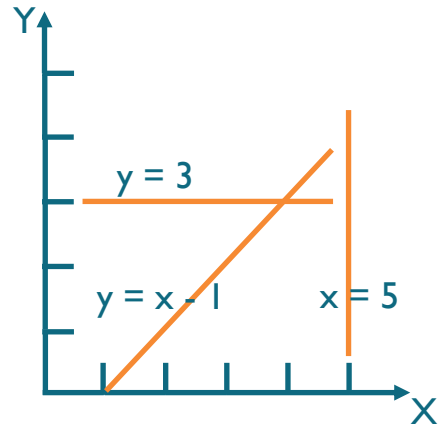



Screenshots from Unity.

# Ray-Box Intersection (I)

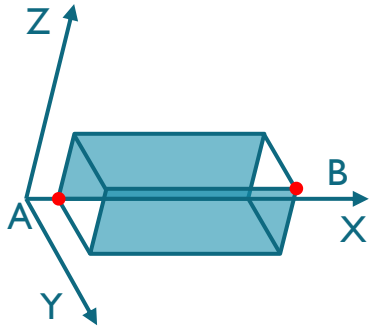
Equation of a line
$y = mx + b$




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

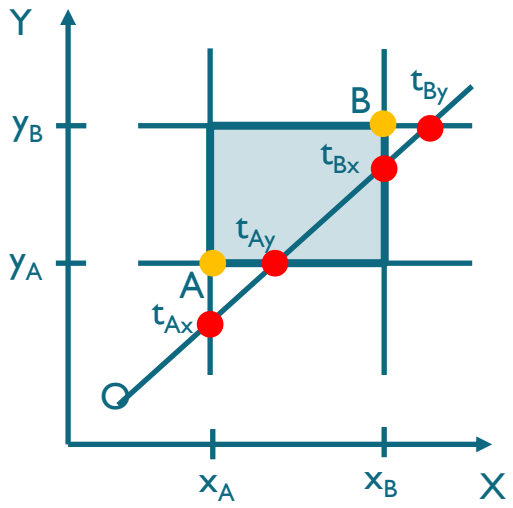
$t_{Bx} = (B_x - O_x) / D_x$
$t_{By} = (B_y - O_y) / D_y$
$t_{Bz} = (B_z - O_z) / D_z$

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.

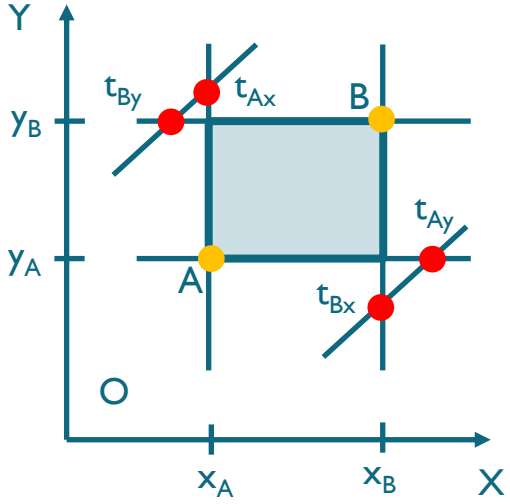
We need the greater value of t parameter for the intersection at the min plane.

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

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

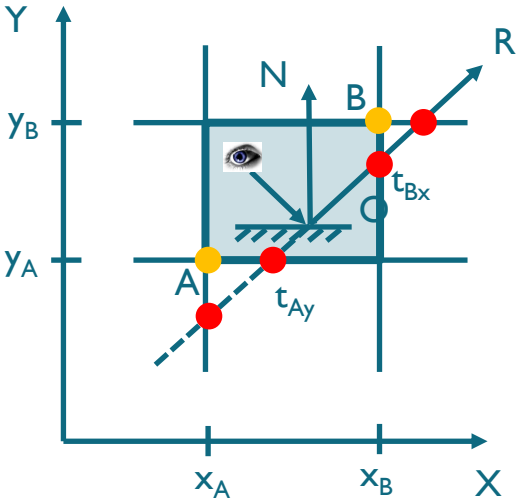
$t_{min} = (t_{Ax} > t_{Ay})\ ?\ t_{Ax} : t_{Ay}$

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 our reflective surface is enclosed by the BBox (i.e the origin of the reflected ray is inside the BBox) then we always have two intersections with the box and the handling of different cases is simplified.



# The Story So Far

- Learned the concept of local cubemaps.
- Learned about a simple and cheap way of simulating realistically environment reflections based on local cubemaps.
- Learned how to implement in the shader reflections  based on local cubemaps.
- Learned how to create, export and import cubemaps in Unity.
- Learned how to apply filters to cubemaps to achieve different visual effects.

# Please Fill  The Feedback Form

# Thanks