The Shadow Rendering Technique Based on Local Cubemaps

Content

This Unity project demonstrates how to implement a new, highly efficient shadow rendering technique based on static local cubemaps.

This project requires Unity 5.0 or higher. It has been tested successfully in Unity 5.0 and Unity 5.3.4.

1. **Importing the project package from the Asset Store**

   Open a new Unity project and from the main menu select *Window->Asset Store* to access the Asset Store from inside the project. Search the Asset Store for "Dynamic Soft Shadows based on Local Cubemap" and press the *Download* button. When the download is completed press the *Import* button. The *Importing Package* window will pop up.

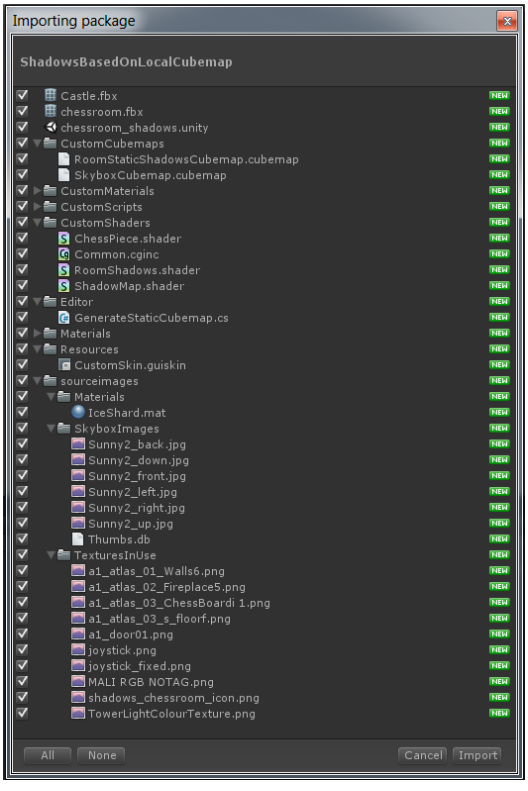

Figure 1. Import package dialog window.

By default all the items are selected. Press the *Import* button to import all the components into your new Unity project.

Under the Assets folder locate the Unity scene file *chessroom_shadows.unity* and double click on it to load it. You will see how the Hierarchy panel is populated with all the game objects of the project.

Press the *Play* button to run the project in the Editor. You must see the shadows on the chess board as shown below:

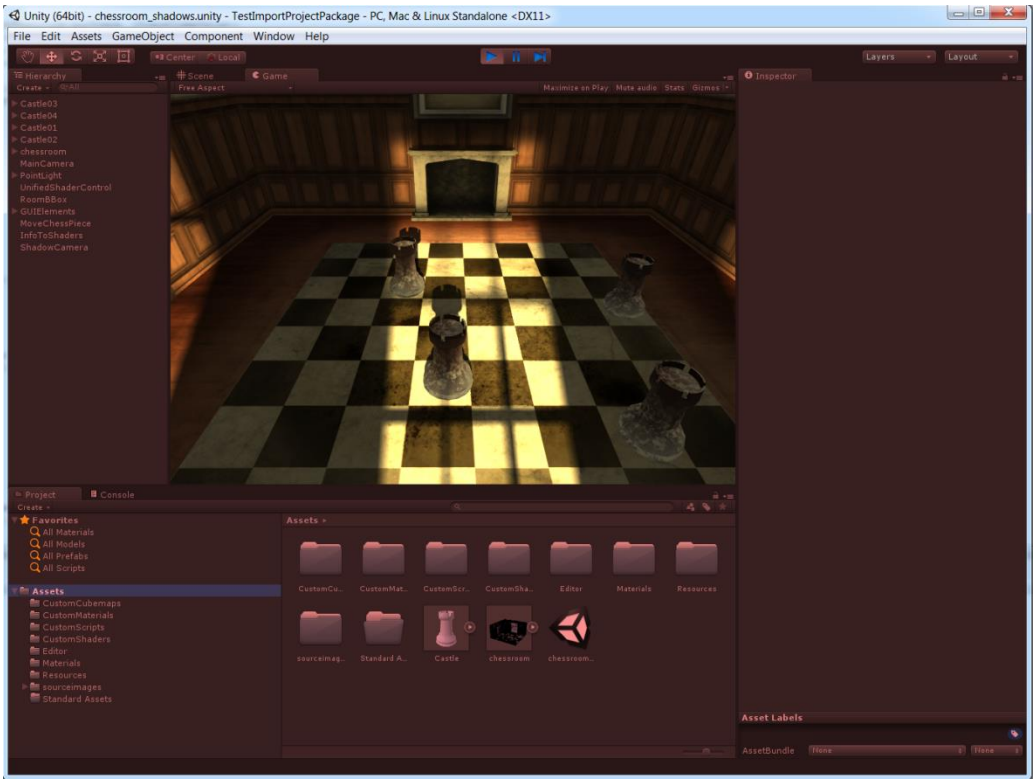

Figure 2. Running the project in Editor.

2. Building the project for Android platform

Build settings for Android have been imported with the project. Connect your Android device to the desktop using the USB cable and Select *File->Build Settings* from the main menu. In the new window select Android platform as shown below and press the *Build and Run* button to build, install and run the application in the Android device. You will be prompted to name the resulting APK.

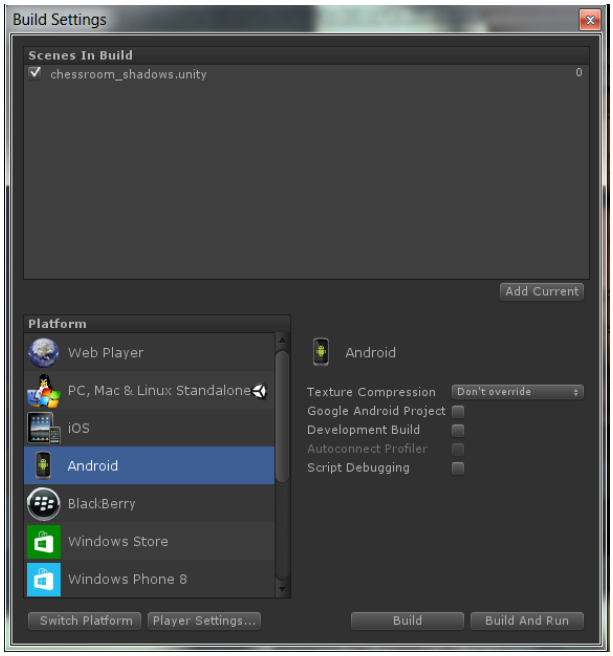

Figure 3. Build settings.

Note that **Developer Mode** must be previously enabled in the device and the device must be properly detected when connected to the USB port.

To enable Developer Mode in the Android device follow the steps below:

- On the device:
- *Settings -> About tablet/phone*
- Tap Build number 8 times
- You should now have an extra option: *Settings -> Developer options*
- Enter *Developer options*, set to On and enable *USB debugging*

Check if the device has been detected using the adb command: *adb devices*

If the device is detected, you should see the following:

List of devices attached WAWJM3SWGY device

If no device is listed as connected then you need to install the ADB driver as it is likely that Windows has failed to find a suitable driver when first connected.
Get the driver from the following website:
http://adbdriver.com/

- Download ADB Driver Installer (Universal)
- Run ADBDriverInstaller.exe

- With the device connected to the Windows machine, press Install
- If prompted, select *Install this driver* anyway

You should now have the drivers installed for this device.

3. How does it work?

This project uses the cubemap *Assets/CustomCubemaps/RoomStaticShadosCubemap* to render the shadows created when the light from *PointLight* game object enters the chess room through the windows. The transparency of the walls of the room has been rendered off-line in the alpha channel of the cubemap.

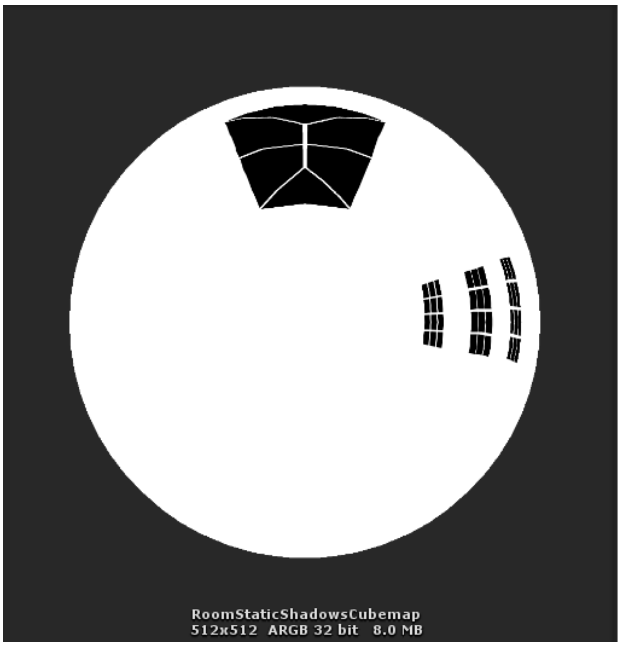

Figure 4. Alpha channel of the cubemap rendered off-line.

The info stored in the alpha channel is like a map of the regions where the light can come through and reach the chess room. In this case the light can come through the windows in the wall and from the skylight.

At runtime, the shaders that implement this technique use the cubemap to render the shadows from the static geometry (windows).

The fragment-to-light vector is used to retrieve the texel from the cubemap. If the retrieved alpha value is zero it means that in the direction of the fragment-to-light vector there is either no geometry or the geometry is totally transparent, i.e. the fragment can be reached by the light and it is lit.

If the retrieved alpha value is 1 it means that in the direction of the fragment-to-light vector there is an opaque geometry, i.e. the fragment cannot be reached by the light and it is shadowed. Any intermediate value of alpha will modulate the intensity of the light that reaches the fragment.

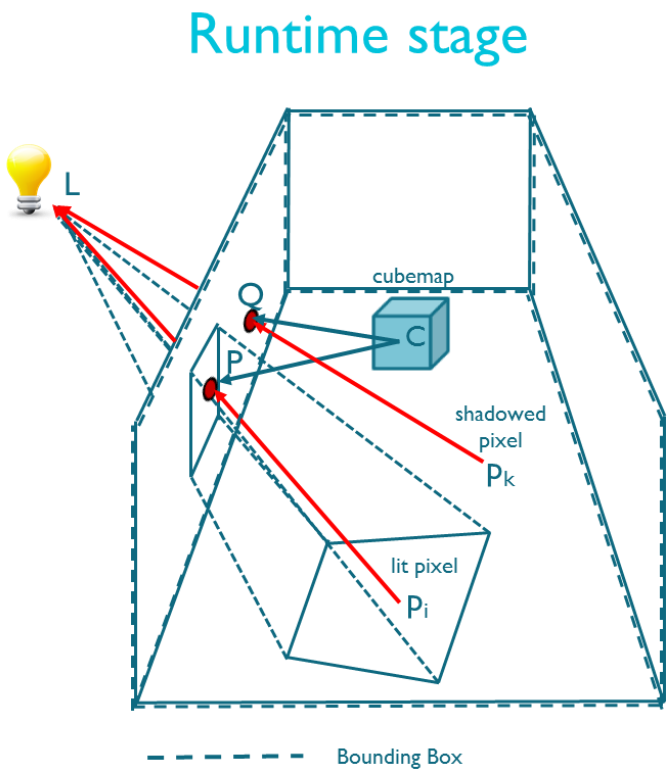

Figure 5. Determination of pixel shadowing at runtime.

Shadows from the static geometry are rendered on the walls, floor, fireplace, and chessboard and also on the chess pieces.

The walls, floor, fireplace and chessboard can also be affected by shadows from the chess pieces. Chess pieces are the dynamic geometry in the scene as they can be moved. Shadows from the chess pieces are rendered every frame using the shadow mapping technique. For the sake of simplicity we are not rendering the shadows from a chess piece on other chess pieces.

Both types of shadows are combined in the shader to achieve a final shadows picture every frame. All shaders are custom shaders.

The materials are located in the *CustomMaterial* folder.

The *ChessBoardMat*, *DoorMat*, *FireplaceMat* and *FloorAndRoofMat* materials use the *roomShadows* shader. This shader renders both types of shadows and combines them.

The *ChessPieceMat* material uses the *ChessPiece* shader. This shader renders the shadows from the windows on the chess piece.

When rendering shadows based on local cubemaps, the first step is to create a vertex-to-light vector in the vertex shader. This vector is then passed as a varying to the fragment shader where the interpolated value is normalized and used to check if the fragment is pointing in the direction of the light source. If so, the local correction is applied to the fragment-to-light vector and used to fetch the texel from the cubemap. The value of the alpha channel tells us if the fragment is lit or shadowed.

The local correction to the fragment-to-light vector is applied using the function *LocalCorrectAndLodDist* defined in the include file *Common.cginc*. The local correction is needed to fetch the texel in the correct direction when working with local cubemaps. This function returns the local corrected fragment-to-light vector and in the fourth component the distance from the fragment to the intersection point of the fragment-to-light vector with the bounding box of the scene. This distance is finally modulated by the factor *_ShadowLodFactor* exposed in the shader's Inspector and used to fetch the texel from the cubemap by means of the function *texCUBElod*. The larger the distance from the fragment to the intersection point the higher cubemap mipmap level used and the shadows will look more blurred. This is how the softness of the shadows is reproduced as this technique makes them more blurred the further they are from the object that creates them.

To perform the local correction, the function *LocalCorrectAndLodDist* needs the minimum and maximum points of the scene bounding box and the position where the cubemap was rendered. This data is passed to the shader as uniforms from the *InfoToShaders.cs* script.

Figure 6. Soft shadows based on local cubemap mipmaps.

You can change the *ShadowLodFactor* at runtime in the Editor using the "*Shadows Lod Factor* " slider in the script attached to *UnifiedShaderControl* game object.

When running on the device, the slide on the top right controls the softness of the shadows while the slider on the top left changes the light position.

Detailed description of this shadow technique and the local correction can be found in the References section.

4. Runtime shadows from dynamic geometry

Shadows from chess pieces are rendered using a camera created at runtime in the *DynamicShadowsCreator.cs* script. The shadow camera is placed in the light position and oriented towards the centre of the chessboard. It only renders the dynamic objects (chess pieces).

The script also creates the texture the camera renders to. This script is attached to the *PointLight* game object. When running in the Editor it is possible to see in the hierarchy the shadow camera

with the name *ShadowCamera*. If it is selected then the Inspector shows that the shadow camera renders to a target texture with the name *ShadowsOnStaticObjs*. By double clicking on it you can see what the texture looks like.

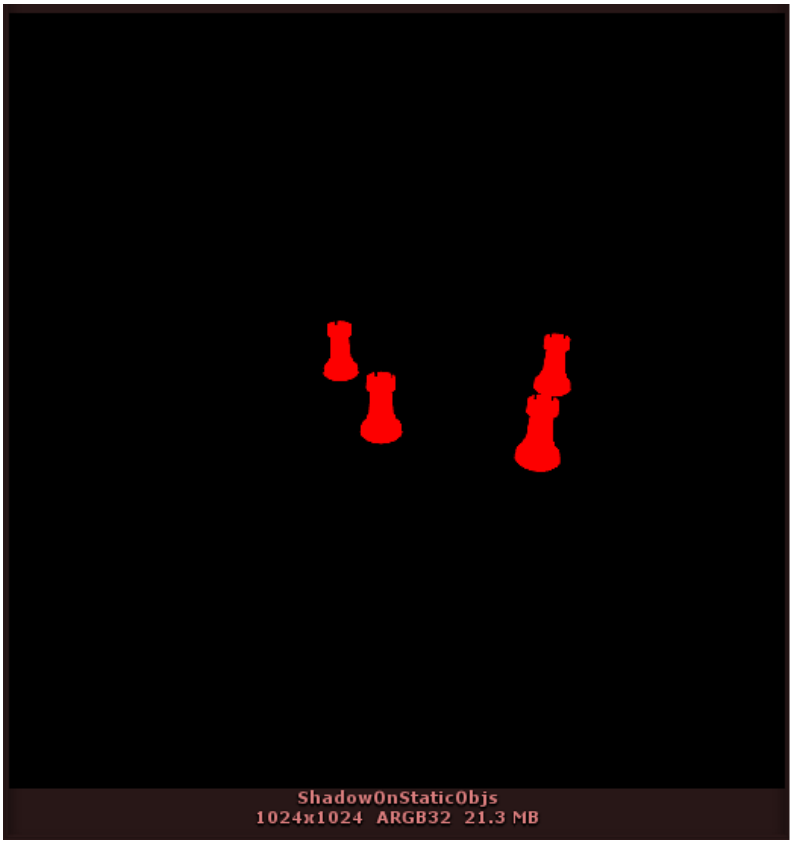

Figure 7. Shadow mapping texture.

The shadows texture is passed as a uniform to the *roomShadows* shader that projects the shadow texture on the geometry of the room in screen coordinates. Finally, the fragment shader combines this texture with the shadow texture from the static geometry based on local cubemaps.

Figure 8. Combined shadows.

5. Rendering the cubemap

The Editor script *GenerateStaticCubemap.sc* renders a suitable cubemap to be used with the shadows rendering technique based on local cubemaps. This tool is available under the main menu *Tools->Render into Cubemap*. Note that the script initializes the variable *cameraBackgroundColor* to *Color(1, 1, 1, 0)* and removes the skybox if there is one. If there is no geometry when rendering the cubemap in this way the alpha value will be zero.

When launching the rendering cubemap the following dialog window is shown:

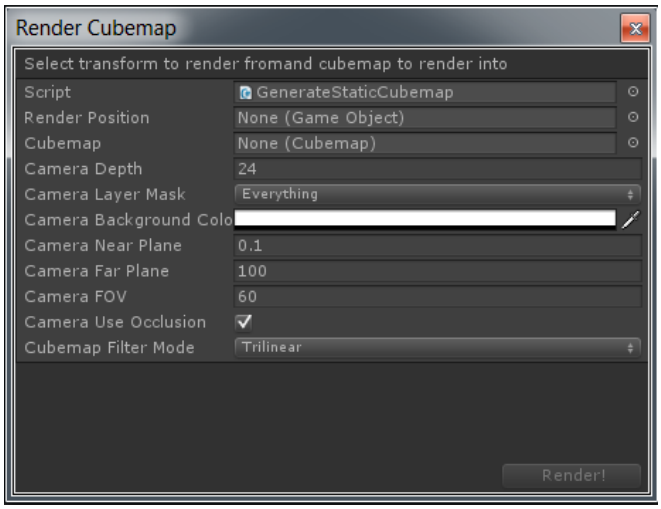

Figure 9. Render cubemap dialog window.

The best rendering position usually is the centre of the scene. Drag and drop the game object *RoomBBox* to this field. Create a new cubemap or reuse the existing one and drag and drop it to the *Cubemap* field. Press the render button.

Set the *Camera Layer Mask* to *StaticObjects* to render only the chess room geometry and exclude the chess pieces.

When creating a new cubemap tick the *MipMaps* check box to allow the cubemap mipmaps to be created which are used for rendering smooth shadows. The *Readable* check box must be also ticked to allow for updating the cubemap during the rendering process.

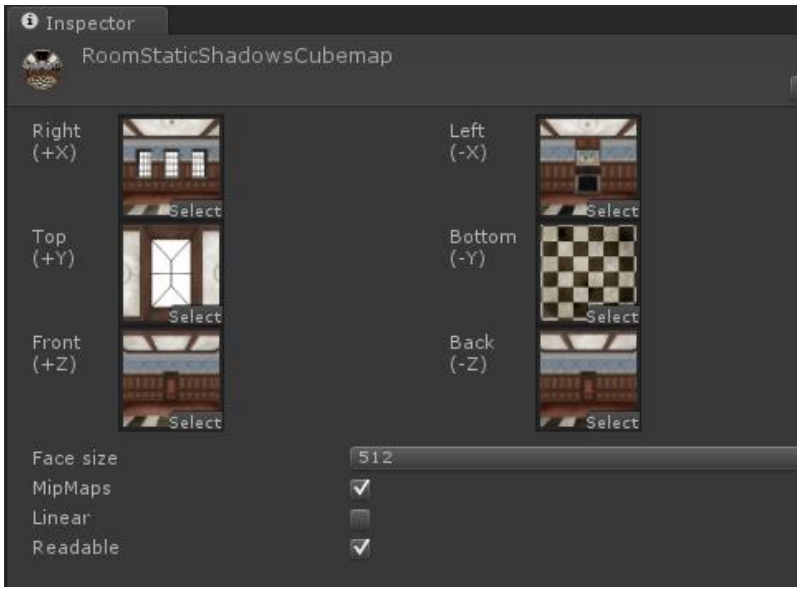

Figure 10. Cubemap settings.

6. Screen resolution

By default the demo will run on the device at 720p screen resolution. The *ScreenResolution* game object under *GUIElements* in the scene hierarchy allows changing the resolution to 1080p.

7. Navigation

The main camera has two scripts attached to it. The *GameCamera.sc* script controls the camera when running on the device. The *FlyingCamera.cs* script controls the camera when running on the Editor.

When running on the Editor use the mouse to point to where the camera is looking and use the keyboard arrow keys and Q,W,A,S,Z,D keys to move.

Use the right and left joysticks to control the camera when running on the device. The right joystick controls the camera orientation and the left joystick moves the camera forward/backward in the direction of the current camera orientation.

8. User Interface

When running on the device the interface has the elements described in the picture below.

When touching the left/right lower areas the joysticks become visible and functional to move the camera.

Chess pieces can be moved by touching and dragging if they are in the central area of the screen. If the chess piece you want to move is not in the central area move and orient the camera accordingly until the chess piece appears in the central area of the screen.

Left slider controls the position of the light.

Right slider controls the softness of the shadows.

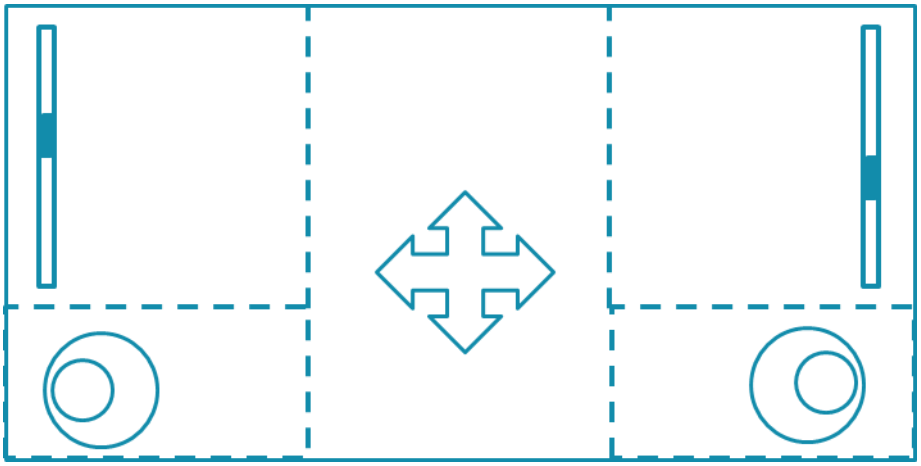

Figure 11. Elements of the interface when running on the device.

9. References

1. *Efficient Soft Shadows Based on Static Local Cubemap*. Sylwester Bala and Roberto Lopez Mendez. GPU Pro 7, Chapter IV Mobile Devices, p.175, 2016.
2. *Dynamic Soft Shadows Based On Local Cubemap*. Sylwester Bala. ARM Connected Community, 2015.
3. *ARM Guide for Unity Developers*. Chapter 6 Advanced Rendering techniques, p. 6-121, 2016.
4. *Enhancing your Unity Mobile Games. Shadows Based On Local Cubemap*. Roberto Lopez Mendez. Workshop delivered at NHTV University, Netherlands, 2015.