# Refract 2D – Table of Contents

# Refract 2D

**Version 1.2**

## 1. **Overview**

Refract 2D is a versatile system for rendering dynamic bumpy surfaces in realtime. The system comprises a set of over 40 hand-written and highly optimized shaders with materials, a Height Map Tool useful for converting Height Maps to Distortion Maps, and an extensive set of example scenes showing off some possible effects.

Refract 2D does not pay attention to perspective, built-in lighting, rotation or shadows. It is optimized for a camera's fixed viewpoint (as in 2D games for example) and provides an alternative image-based lighting system. Although ideal for 2D purposes, there are many ways to use it in a 3D environment where the lack of true three-dimensional depth is not noticeable.

Feel free to play with the example scenes right away to get a feel for what Refract 2D is capable of and to see some possible uses. The scenes are merely examples and represent only a handful of many possible effects achievable with the system. Refract 2D's versatility lies in your freedom to use up to 4 Distortion Maps, up to 3 textures, adjustable combinations of refraction, reflection, and image-based lighting, adjustable dynamically-combined distortions, and efficient rendering, all within a single optimized shader hand-built for the job.

Whether you're looking to implement refraction effects, reflection effects, optimized lighting, image distortions, bumpy environments, surface animation, or to just have fun, Refract 2D shines.

## 2. Workflow

Refract 2D features a simple and flexible workflow. The typical workflow is as follows:

1) Create one or more Height Map textures/backgrounds

2) Process the Height Map textures with the Height Map Tool to produce Distortion Maps

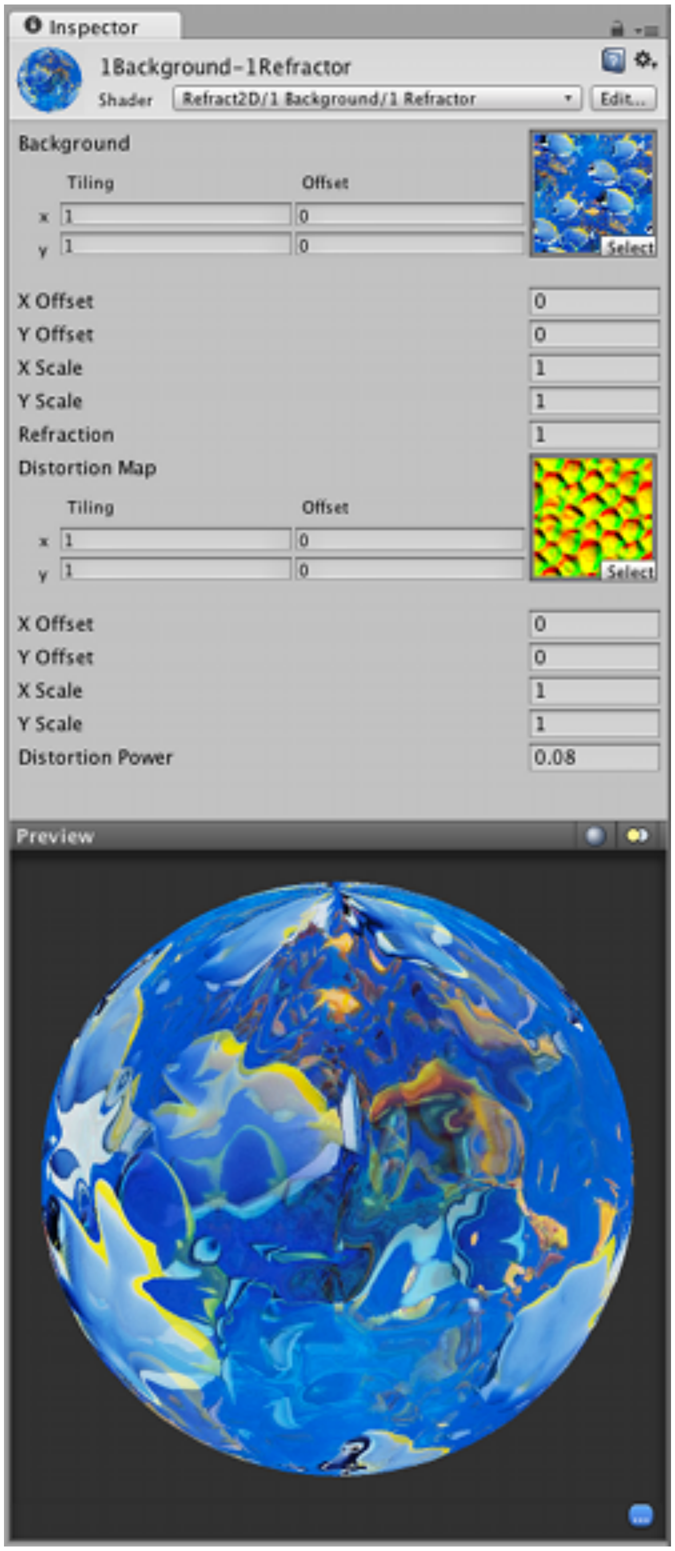
3) Create a material with one of Refract 2D's shaders chosen

4) Assign the Distortion Map texture(s) to the material

5) Create background, foreground, midground and/or lightmap textures

6) Assign those textures to the material

7) Adjust the material's settings to get the effect you want



1Background-1Refractor

Shader: Refract2D/1 Background/1 Refractor

Background

| | Tiling | Offset |
|---|---|---|
| x | 1 | 0 |
| y | 1 | 0 |

| | |
|---|---|
| X Offset | 0 |
| Y Offset | 0 |
| X Scale | 1 |
| Y Scale | 1 |
| Refraction | 1 |

Distortion Map

| | Tiling | Offset |
|---|---|---|
| x | 1 | 0 |
| y | 1 | 0 |

| | |
|---|---|
| X Offset | 0 |
| Y Offset | 0 |
| X Scale | 1 |
| Y Scale | 1 |
| Distortion Power | 0.08 |

## 3. Alternative Workflows

Provided you supply Refract 2D's shaders with appropriate textures, it really doesn't mind how they were created. Its only requirement is that the Distortion Map(s) contain appropriate data in the Red and Green channels. This means you can create Distortion Maps using methods other than the Height Map Tool. This could include computing a map using your own scripts, generating a map from other software, using image-processing techniques, rendering the map in realtime to a Render Texture, or even hand-drawing the texture data. You can even use a regular image texture or Height Map texture for a Distortion Map - it will still work for quick results but may or may not provide the effect you want.

## 4. Height Maps

In Refract 2D, a Height Map is simply a grayscale image representing the height of the surface. It can be stored in either  Alpha8, RGB24, RGBA32 or ARGB32 texture formats.

Compressed textures or textures in other formats are not supported as inputs to the Height Map Tool. The primary purpose of the Height Map is to pass surface information into the Height Map Tool, which it uses to generate a Distortion Map. You typically will use Distortion Maps with the Refract 2D shaders.

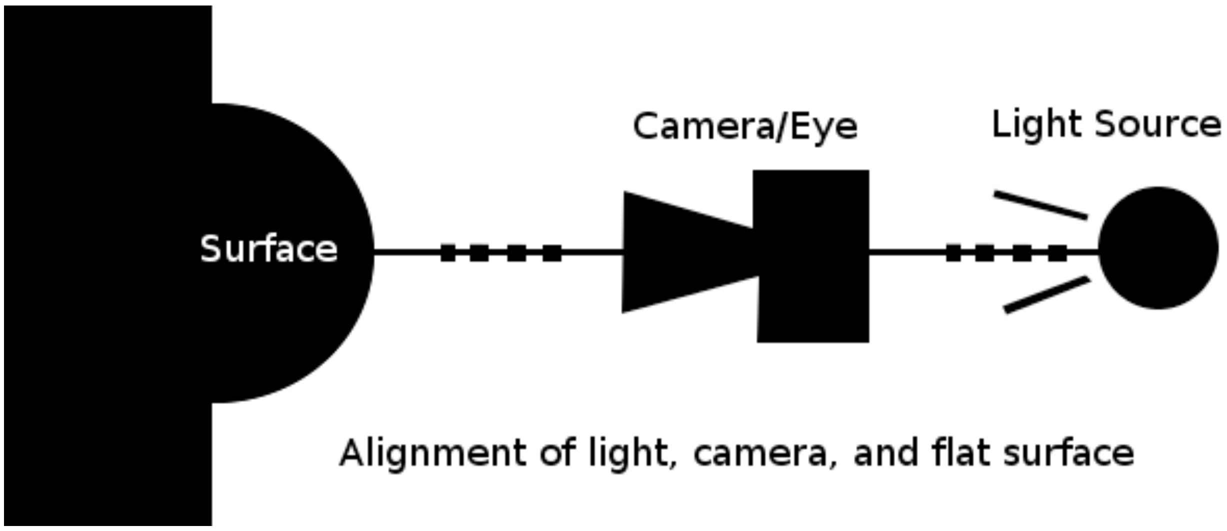
The Height Map Tool will obtain height information from either the Red channel of your grayscale texture, or the Alpha channel in the case of Alpha8 textures. Your texture generally will be grayscale but really only the red channel is needed. If you are tight on space, I recommend using Alpha8 format.

A Height Map texture can be any size, but bear in mind that only one set of UV's are shared by all of the textures used in a Refract 2D shader. Typically square textures are faster to render than rectangular. You can still define the UV coordinates of the vertices in your mesh to render alternative-shaped areas.

The Height Map's pixel values are used to represent height, where 0 is the lowest point on the surface and 255 is the highest point. This may also be thought of as a 0..1 range. It is trivial to use an image of some pebbles as a Height Map, for example, where the brightness of the pebble surface may determine its shape. Black typically represents a low surface while White typically represents a high surface. Alternatively, use black for a low surface and bright Red for a high surface.

The purpose of the Height Map is to *indirectly* model the slope of the surface. You can infer what the slope will be based on the difference in value between adjacent pixels. For example a pixel with a value of 50 followed by a pixel of value 200 implies a difference in value of 150. This value then directly maps to a given `surface angle` or `normal`. While you don't directly supply surface normals in the Height Map, you can model them based on the *relative* difference between pixels.

**4.1 The Ideal Height Map**



Usually when you use a source image, for example a photo of some pebbles, it will be a photograph of a 3D object lit by a light source. Ideally the light source should be directly in front of or behind the camera's straight-on view of the object. If you use photographs whose lighting is off to one side, you will find it results in a Height Map that causes the resulting lens effects to be distorted to one side. The Height Map Tool does not attempt to interpret the original direction the light was shining from and assumes it was aimed from directly in front of or behind the viewer `square-on` toward a `flat` surface.

The light in such an image will naturally shade the pixel values to represent the direction/intensity that light would reflect off the surface into the camera (not taking into account any radiosity/reflections/shadows etc). This ultimately provides an impression of surface shape. Refract 2D's Height Map Tool will reverse the process, inferring the direction of the surface that the light bounced off based on the amount of illumination in the pixel. Bear in mind that this ideally applies to completely smooth, featureless surfaces. Fluctuations in surface color, texture, reflections, shadows and indirect light will be interpreted as fluctuations in physical shape. For example a small fleck of texture will be translated to mean the surface was slightly different at that point and thus reflected more light.

## 4.2 Smooth versus Rough

Give consideration to the smoothness that you want your effect to have. Without applying any smoothing or blurring to a Height Map image, subtle fluctuations in pixel color will translate into fine details in the lenses, which might be suitable for a detailed frosted-glass effect but not a smooth magnifying-glass lens. Experiment with `blur` effects in your favorite graphics software to obtain the level of smoothness you desire from your surface. The Height Map Tool does include the ability to apply smoothing, but you will find it is likely slower and of poorer quality than that provided by dedicated graphics software.

In general, applying a Gaussian Blur to your image will smooth it out and provide smoother surfaces for Refract 2D to use. Feel free to use any other image-processing techniques, graphical effects, or other sources of pixel data to build a Height Map for the Height Map Tool to use.

## 4.3 Clamped versus Wrapped

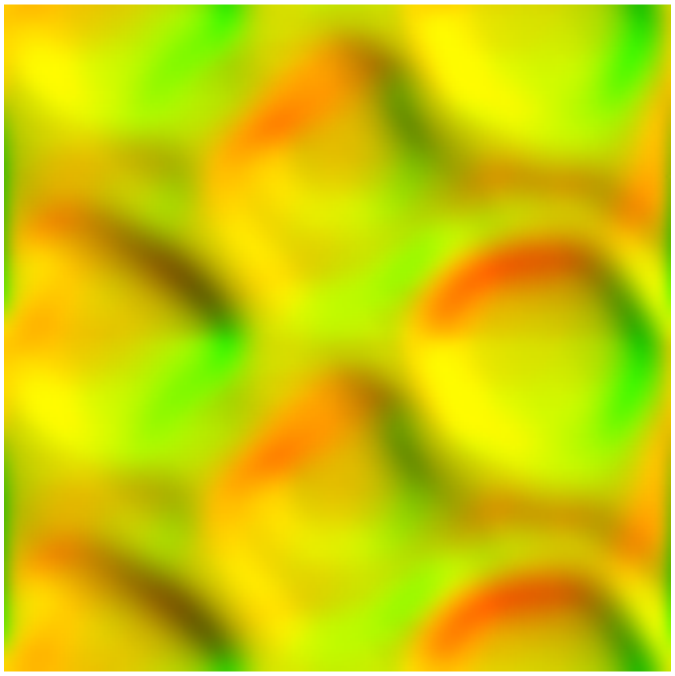
Give some thought also to whether your Height Map needs to scroll, or to put it another way, whether the Distortion Map you generate from it will need to scroll and wrap around seamlessly as if it has no edges. The Height Map Tool can attempt to treat the generated Distortion Map as though it either wraps or clamps in the X and Y directions, but to obtain a true wrap-around experience you will likely need to prepare your Height Map as a seamless texture. There are several seamless texture tools available, just try searching online. Gimp is also a free graphics tool which includes a seamless texture operation. Paint Shop Pro also can make an image seamless with a little more control and there are dedicated software apps designed for this very task.

Using an improperly prepared (ie clamped) texture and expecting it to wrap around a 3D object or to be seamlessly scrollable will result in visible `edge` or `bumpy line` in your Distortion Map. An alternative option is to generate your Distortion Map using a clamped texture, and then apply a seamless process to the Distortion Map texture. Even though the Distortion Map contains `data`, there is nothing to stop you applying image-processing operations to it after it has been generated. For example you can apply an additional blur, make it seamless, sharpen it, interpolate it, resize it, distort it, or any number of other transformations and the data will still generally work with the shaders.

The only area to be careful when adjusting Distortion Map textures is with regards to pixel brightness. A flat surface is represented by `mid gray` values of 128 in a 0..255

range, or 0.5 in 0..1 range. Values of 0..127 or 0..0.5 represent negative X and Y slopes while 128..255 or 0.5..1 represent positive X and Y slopes. You ideally want to keep your Distortion Map `centered` in the range of values. Skewing the pixel values brighter or darker will distort the surface shape, which may or may not be desirable. Interpolating pixel values (for example by scaling-up the Distortion Map) is perfectly acceptable.
If you are creating a Height Map texture from a photograph, generally give thought to how the brightness levels in the image will translate to the shape of an effect. Sometimes you might need to find a photo which was taken at a different camera angle rather than `straight on` in order to get the effect you want. You may also find that images you thought might produce a good distortion don't work so well. Some experimentation is often needed and it can be actually fun to discover the possibilities presented by different Height Maps.

## 5. Distortion Maps



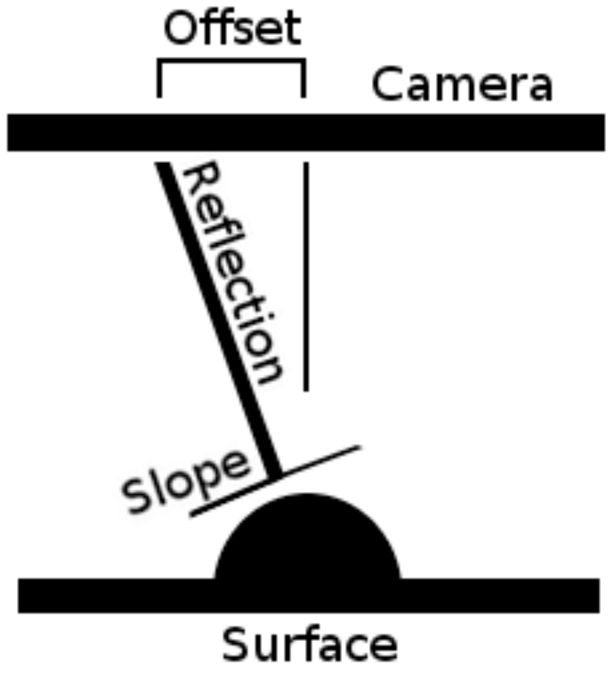
In Refract 2D, a bumpy surface is converted from a Height Map to a `Distortion Map` using the Height Map Tool. The Distortion Map models the bumpiness or shape of the surface and how to use that information to render the distortion in realtime.

Whereas a Height Map provides the absolute height of the surface, a Distortion Map stores information about the relative horizontal and vertical slope of the surface. Actually it stores offsets (not angles) which represent where light leaving the surface's slope would collide with a flat plane of a background or foreground texture.

A Distortion Map will appear mostly yellowish with some green and red parts. This is because the Red and Green channels of the texture store displacement data derived from the surface slopes. As mentioned in the Height Map section, a color of 128,128,0 (mid yellow) will typically represent the flat parts of your surface. Changes in the Red component reflect changes in the horizontal slope of the surface, while changes in the Green component reflect changes in the vertical slope. Lower Red or Green values represent negative slopes (aimed toward upper left) while higher values represent positive slopes (aimed toward lower right).

A Distortion Map generally can be considered invisible in your scene, like a glass lens. A surface can be modeled by a Distortion Map with per-pixel accuracy, where every pixel represents a different contour or lens fragment. This provides complete flexibility in the kind of surfaces that can be represented by the Distortion Map, with much more precision than simply modeling the surface using triangles. A highly complex distorted surface can be stored in a single Distortion Map and rendered in a single draw call. Up to 4 Distortion Maps can also be combined and merged in realtime inside a single shader.

### 5.1 How the Distortion Works



Distortion Maps differ from traditional Normal Maps in that they not only represent the surface slope, but also go a step further. The Height Map Tool will calculate the slope of the surface based on its brightness, and then *use* the surface normals, performing part of the process that would normally run in the shader, to trace rays of light to the camera. The tool will trace a ray from each surface slope toward a flat plane which lies parallel to the surface, representing the camera (or a flat 2D background, environment map, light map, etc).

The calculation of where reflected light would hit the parallel surface is performed offline in the Height Map Tool. In a way, this is like `baking` the normal map based on a fixed camera angle and thus converting it to un-rotatable `object/world space`. This is perfectly acceptable for a 2D scenario where the camera does not rotate around the X or Y axis and provides a performance boost in the shaders. Distortion Maps may differ slightly from object/world-space normal maps (which are also red/green/yellow) in that they store offsets, not angles.

Having found the point on the camera/background plane where a ray from the surface collides, The Height Map Tool also takes it a step further by converting this information into horizontal and vertical `offsets` from the center. The further away from the center (directly above a flat surface slope) the intersection point is the greater the horizontal and/or vertical offset values. These offsets are then `baked` into the Distortion Map texture as X-Y offsets in the Red and Green channels. Their value and range depends on the shape of the surface and various settings in the Height Map Tool such as the contrast adjustment. Later in the shaders, these pre-calculated 2D offsets can simply and efficiently be added to the UV/texture coordinates from the mesh, providing a realtime per-pixel displacement of the textures. The result is that the textures will appear to be distorted correctly, and dynamically, based on the shape of the original surface.

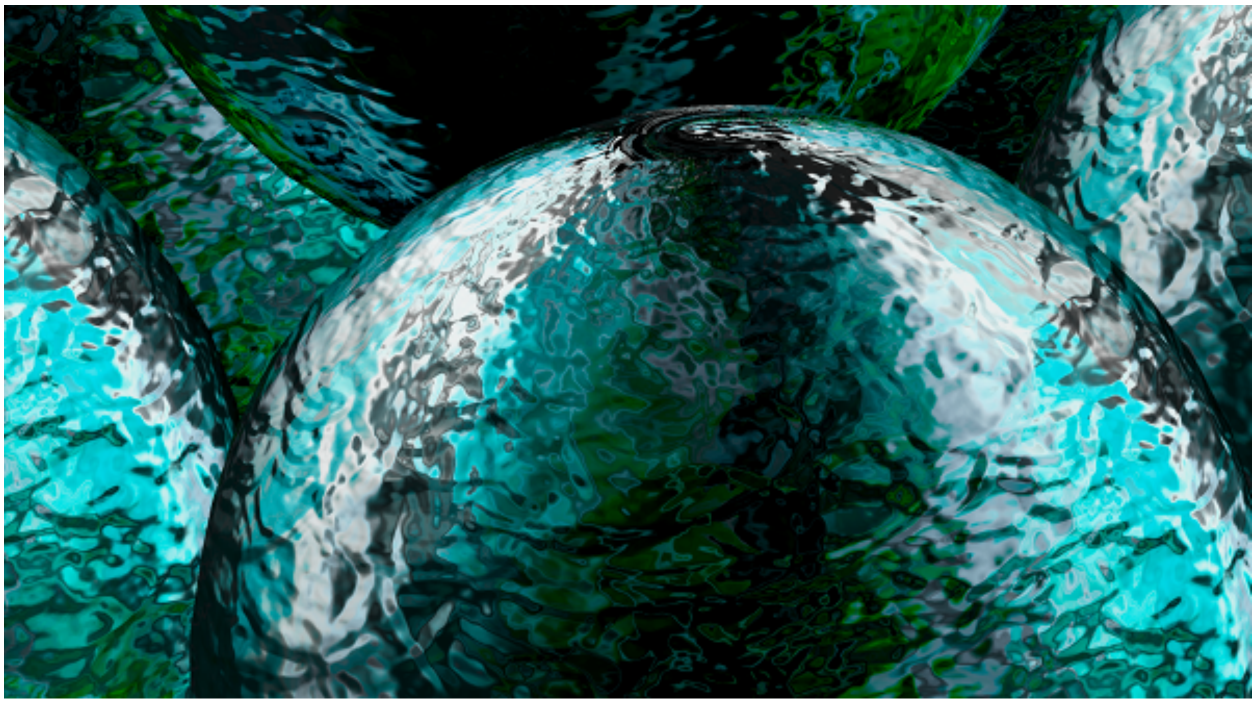
Although the distortion effect is pre-computed with a camera always looking directly at the surface, the position, scale, `distorting Power`, and degree of refraction vs reflection can be modified for each Distortion Map in realtime. This allows a texture to be warped dynamically. Additionally, even though Distortion Maps are pre-calculated, the shader is able to combine up to 4 of them cheaply in realtime before applying the final distortion to image textures. This allows up to 4 `layers` of lenses to be combined into one virtual lens, in a way that still provides realtime refraction, reflection and animation.

This provides versatility and offers interesting animation opportunities. You can merge multiple distortions into one effect - not just statically but with each Distortion Map being individually adjustable in its contribution. You could cross-fade between distortions, or transition from refracting to reflecting for a given Distortion Map. You can also increase or decrease the distorting `power` of a Distortion Map by adjusting its Power value in the material. A low value of 0.08 is fairly normal, so for example 0.16 doubles the power of the lens (without changing its shape), emulating materials which have a different refractive strength given the same shape of lens. It is possible to go widely outside normal laws of physics by allowing a Distortion Map to be grossly over or under-powered in its distorting strength.

Typically a normal surface would probably not provide a wide `spread` of light rays. In other words, looking at a background through it would bring in light rays from a fairly wide but limited area - you would perhaps start to see pixels from outside the lens area showing up `within` the lens. However, Refract 2D allows you to magnify the lens power to provide texture look-ups over a huge area - provided your background textures are set to `Repeat` instead of `Clamp`, or are large enough, the lens will happily reference far-off texture locations which wrap around the texture edges many times. Additionally each background texture has its own individual `Refraction` value which allows each texture to model the distortion differently. This can be useful for example, to overly distort a texture representing lighting (an image-based `light map`), which can result in an intensified impression of surface shape based on light and shadow.

## 5.2 Distortion Maps in 3D Scenes

Even though the pre-computed camera is fixed and does not take perspective or rotations into account, Distortion Maps can still be used in 3D scenes provided you can accept that the distortion applies against a `flat environment` parallel to the surface of the 3D object. In other words, in 3D, the Distortion Map simply shrink-wraps onto the surface of the object and acts as though the `camera plane` similarly shrink-wraps to match. Applying a Distortion Map shader to a 3D surface will *possibly* look okay, given that the resulting effect will wrap around the object. However, the effect will be treated as though it were actually `flat` and not bumpy. Perspective and rotation will still be applied to the shader output, which to a degree will make it look 3D. But the individual bumps will not be technically placed or lit correctly given the angle to the real camera.



The surface may still look bumpy, however, and it’s possible that there isn’t enough obvious visual error for a viewer to realize the inaccuracies, especially if the surface distortion is animated. A certain amount of natural `compression` occurs at the sharper-angled edges of 3D objects, for example at the sides of the sphere above, creating the impression that the effect is truly 3D.

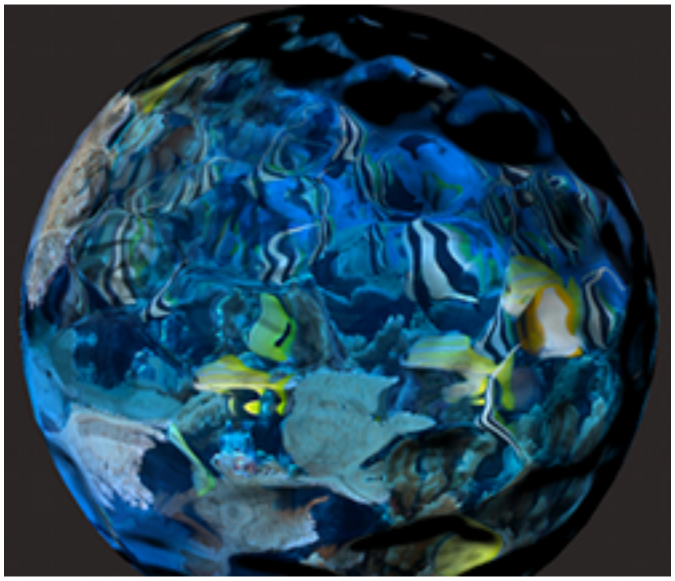
Unlike parallax mapping and other methods which give surface features a genuine 3D-perspective viewpoint, Distortion Maps act more like typical normal maps. The difference between them and normal maps, however, is that normal maps can deal with moving light sources from any angle, whereas Distortion Maps model light based on image data and/or light maps. Also Distortion Maps may refract/reflect a significantly wider range of texture data, resulting in bigger bumps. Similar to normal maps, however, Distortion Maps viewed in 3D will only render inside the silhouette of the geometry, so will not create a bumpy profile.

Another thing to bear in mind is the direction the viewer is looking at the world, and how this influences what would be refracted or reflected. Remember that Refract 2D is essentially a `flat` bump-mapping system and uses a fixed camera view. Therefore for example, if you were looking at a wall segment of a corridor in perspective, a refractive surface should refract light from objects `behind it` in terms of what `behind` means from the viewer’s vantage point, but will instead show refractions perpendicular to the surface of the object, off to the side. Some clever use of Render Textures to dynamically capture the background that are actually `behind` the surface could possible alleviate this issue. The case is similar for reflections. Refract 2D is good for `faking` local reflections and refractions in 3D, but not so good for true 3D effects.

Bear in mind also that a Distortion Map is a texture with a fixed resolution. This may provide perfect pixel-to-texel mapping in 2D, but in a 3D-perspective environment it is possible to move closer to a surface and not experience increased detail. That said, and especially if your textures use bilinear or trilinear filtering, Distortion Map data will be accurately interpolated to still provide high-resolution smooth results even when close to an object. You may be surprised how low-resolution your Distortion Maps can actually be and still maintain high quality in the final output - this is partly due to how a bumpy surface re-samples the texture in realtime, effectively increasing the resolution of the surface.

It is okay to use anisotropic filtering, bilinear filtering, trilinear filtering and mipmaps with your Distortion Map textures and other textures for best results. Offset data in the Distortion Map will simply be interpolated or sampled appropriately and still produce correct results.

### 5.3 Interpretations of `Distortion`



A Distortion Map can be used in different ways, since it only models the surface shape and not the rest of the environment surrounding it.

- It can be used to view a background positioned further in the distance whose light is refracted through a bumpy surface - in this case the Distortion Map can be thought of as a glass lens.

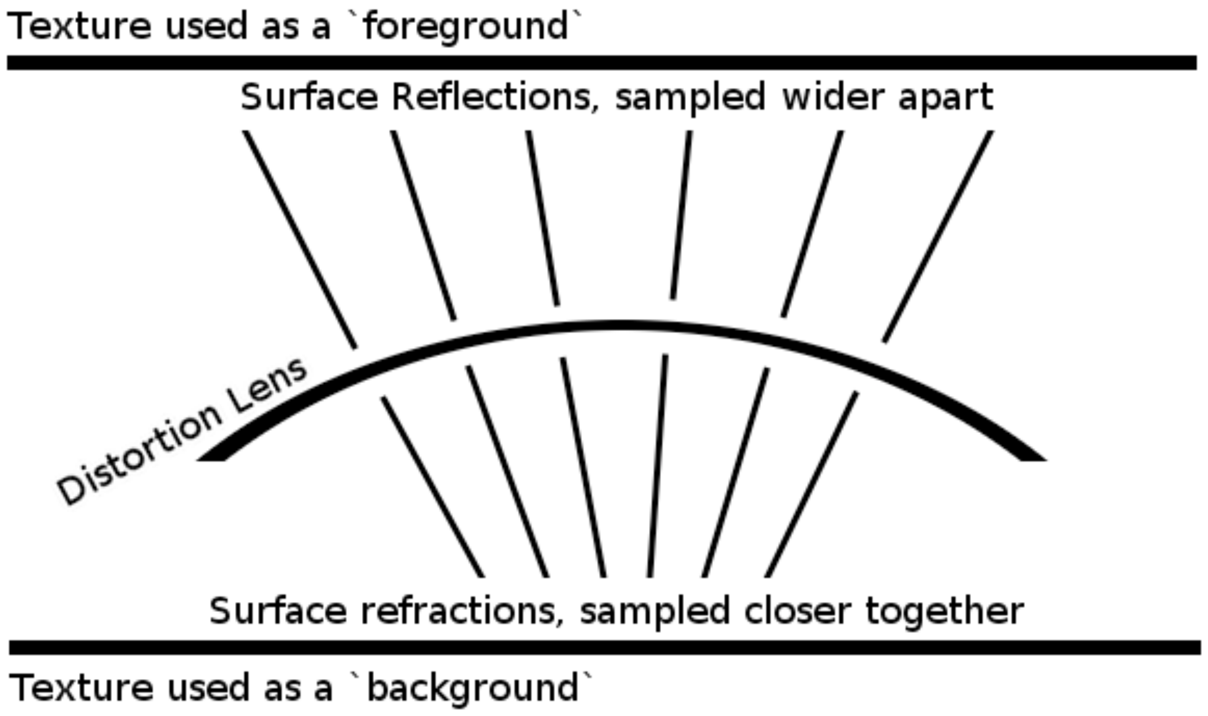
- It can also be used to view a nearer foreground environment as seen in reflections off the surface - in this case the Distortion Map can be thought of as a shiny reflector.

- Additionally it can be used to access a static or realtime-generated light map in the form of image-based lighting - in this case the Distortion Map can be thought of as a way to model the reflection of light from multiple light sources. Each of these interpretations can be used independently per texture or in combination.

For example, you can use a single Distortion Map with a single `background` texture to model refraction of the background as if seen through a lens or bumpy glass. Alternatively you could use the same Distortion Map with a single `foreground` texture to reflect an environment map. Also you can create dynamic lighting effects by including a `light map` texture containing a cross-section of pre-rendered per-pixel lighting.

You could optionally generate or modify your texture data in realtime using Render Textures. This would allow you to, for example, draw and move dynamic light sources as the player scrolls through an environment. It would also allow you to pre-render moving objects and then use that image as an environment to perform realtime reflections.

Refract 2D's `backgrounds` are initially arranged under the assumption that you would likely use one refracted background, one reflected environment map, and one even closer light map. However, these assumptions are not fixed and each `background` texture can be used for refraction or reflection independently. Each shader tends to refer to the image textures as `backgrounds`, but they can be thought of as either behind or in front of the lens. It is whether the image is set up with a positive Refraction value to represent `refract as a background`, or a negative Refraction value to represent `reflect as a foreground`. The image below demonstrates how a single Distortion Map lens may represent a surface used for both refraction and reflection.


Texture used as a `foreground`
Surface Reflections, sampled wider apart
Distortion Lens
Surface refractions, sampled closer together
Texture used as a `background`

At the center of a refracting Distortion Map bump, samples are closer together, which results in an apparent `magnification` of the texture like from a magnifying glass. When the Distortion Map is used for reflections however, the texture is sampled further apart resulting in what may appear to be a shrinking of the texture, yet reflecting it from all directions. Refract 2D allows you to transition between refracting and reflecting behavior for each texture individually in realtime, as well as adjusting each individual Distortion Map between refracting and reflecting behavior. By default, Distortion Maps refract the light with a Refraction value of 1.

Altering the Refraction parameter in the Material, or the Power parameter, in a transition from positive to negative values, allows you to effectively `bow` the lens to change it from a refractive lens to a reflective surface. Indeed, a Refraction value of 0 results in no distortion at all, which can be used to include a background layer that you don't want to distort or to temporarily take a Distortion Map out of action while distortion from another Distortion Map becomes dominant.

# 6. Background Textures

In Refract 2D you can use up to 3 backgrounds plus an optional mid-ground as part of the shader. While the backgrounds are referred to as a `background`, they can really be thought of as just a layer, which could be used for a background or a foreground.

A background is just a texture containing whatever image you want to distort or reflect - it could be a surface texture, a photograph, a distant background, a foreground, or something representing the light in the scene. It could be a static texture, something you generated, or something you just rendered to a RenderTexture. Refract 2D will accept any texture format for background textures, although typically it will pull from the Red/Green/Blue channels for color values so you might not want to use Alpha8 format (unless you want to only use the alpha channel). Refract 2D will use the alpha channel of the first background texture and/or an optional mid-ground texture only.

The texture does not have to include an alpha channel, and in many cases the alpha channel of the texture is not used. However, in the case of shaders featuring 2 or 3 backgrounds, the alpha channel of the first background is used to alpha-blend between the other backgrounds. Take a look at the `Water` example scene for a good example of this in use. In the example, the main `water` background is the first background and contains an alpha channel to mask out the sky. The second background is a sky texture with a Refraction value of 0, so it appears not to distort. And the third background is a lightmap with white pixels covering the sky area, so that when the lightmap is multiplied by the sky texture the sky texture comes through in full brightness.

Handling of backgrounds, blending and alpha channels differs based on the number of backgrounds present:

- **With 1 background:** It simply is the only background so does not merge with other backgrounds.

- **With 2 backgrounds:** The two backgrounds are cross-faded, ie alpha-blended, based on a combination of the alpha values from the first background and the \`Fade\` parameter from the second background. This in a way assumes you might have a foreground \`environment map\` which is reflected and a background that's refracted.

- **With 3 backgrounds:** The combining of 3 layers is a little more complex. Firstly an alpha-blend cross-fade is performed between the first two backgrounds, as for the 2-background shaders. The amount of blending is based on the Fade value of the second background multiplied by the alpha channel from the first background. This produces a cross-fade of the first two backgrounds. The result of this is then alpha-blended against the third background, using the third background's Fade value to transition between the third background and the result of the first alpha-blend. Here the third background might represent a light map.

## 6.1 The Mid-Ground Texture

Some versions of the shaders also support a \`mid ground\` texture. This can be used to overlay non-distorting pixels on top distorting pixels, possibly in-between a refracted background and a reflected foreground. The mid-ground is not affected by any distortions. The mid-ground might represent the main play-field of your 2D game where your character moves around, or aspects of scenery that you wish to overlay on top of your distortion effect to mask it.

To a degree, the mid-ground can be used to give \`color\` to your Distortion Map, suggesting that it doesn't let though or reflect all light, but instead only allows light of a given color. For example black/full-alpha pixels in the mid-ground will completely block all light coming through the Distortion Map. To achieve stained glass windows for example, the mid-ground can provide some color by only allowing light of a given color value.

This somewhat models the idea that a transparent lens may contain some particles of pigmentation which filter the light and thus block certain frequencies. The more color you add to the Distortion Map itself through the mid-ground texture, the more it blocks the light, and therefore less distortions are seen through it. This is why the mid-ground alpha-blends against the distorted background, to simulate the blockage of light and the presence of color in the surface itself. You could also use this as a way to define the glossiness or shininess of the surface, modulating how reflective or refractive it is on a per-pixel basis and how much of its own color should show through.

It can also be thought of simply as a way to mask a shape around a distorted area, which can be useful for wrap-around animations (for example scrolling the Distortion Map vertically) to fade out the distortion effect toward the top of the area so that it blends into the background.

The mid-ground's contribution to the overall effect varies slightly based on how many backgrounds are used:

- **With 1 background:** With only one other background to consider, the alpha channel from the mid-ground texture is multiplied by the mid-ground's Fade value, and this is then used to alpha-blend between the mid-ground and the first background. The alpha channel of the first background is ignored.

- **With 2 backgrounds:** The two backgrounds are alpha-blended based on the alpha channel from the first background multiplied by the Fade value of the second background. Then the mid-ground is laid on top by alpha-blending it against the previous result, based on the alpha channel of the mid-ground multiplied by the mid-ground's Fade value.

- **With 3 backgrounds:** The combining of 3 layers is a little more complex. Firstly an alpha-blend cross-fade is performed between the first two backgrounds, as for the 2-background shaders. The amount of blending is based on the Fade value of the second background multiplied by the alpha channel from the first background. This produces a cross-fade of the first two backgrounds. The result of this is then alpha-blended with the mid-ground texture based on the alpha channel and Fade value of the mid-ground. Finally, the result of this is then alpha-blended against the third background, using the third background's Fade value to transition between the third background and the result of the previous step.

## 6.2 Multiplied and Added Versions

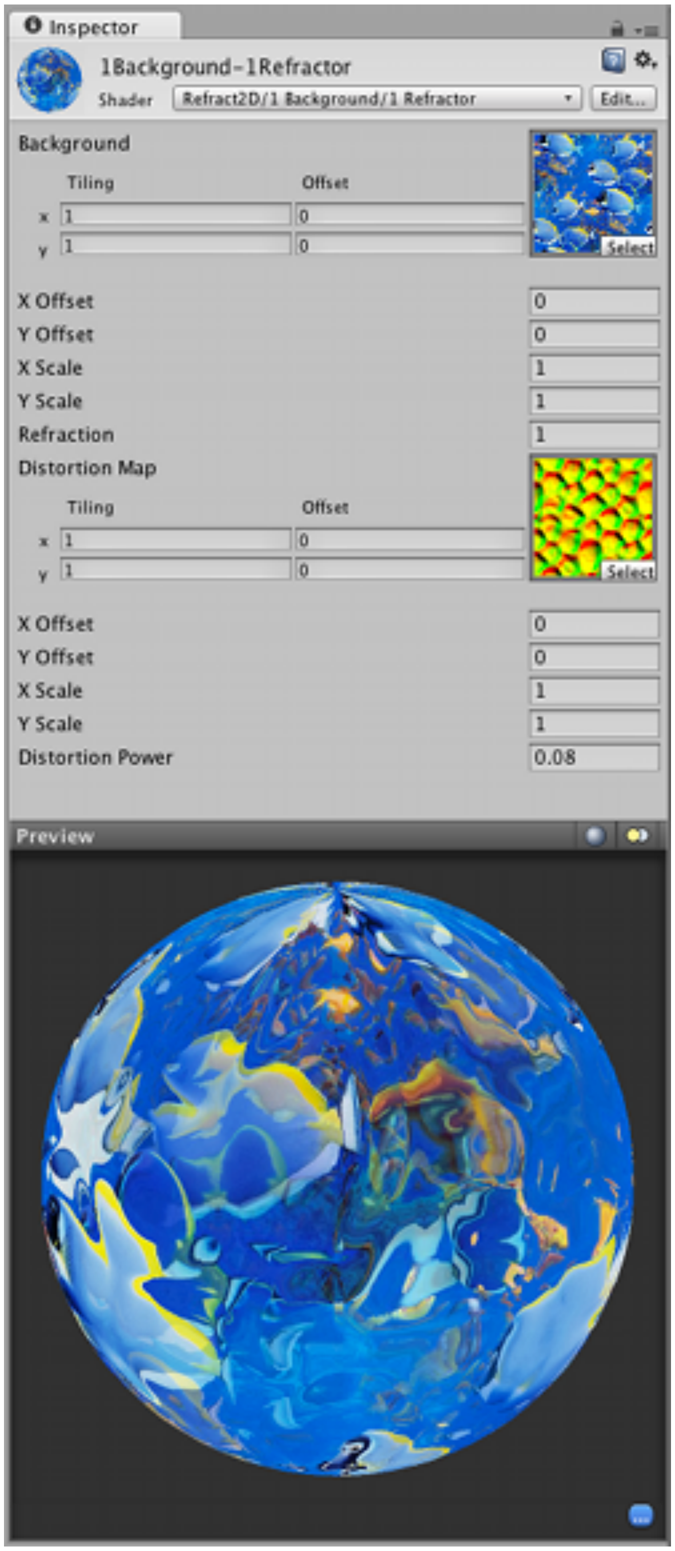
Each of the shaders featuring a mid-ground also typically provide a Multiplied and an Added version. Since normally the mid-ground is part of an alpha-blend operation, in the Multiplied version the mid-ground is instead multiplied against the previous layers. Alternatively the mid-ground can be Added to the previous layers. These shaders offer some alternative results in cases where you might want to use the mid-ground to provide non-distorted lighting, for example, or where you want to multiply or add a texture on top of the effect.

# 7. Shaders and Materials

Refract 2D features more than 40 hand-written shaders. The shaders have been highly optimized by hand to use the absolute fastest data formats possible. They are also fairly well commented. Texture data is read using the `fixed` variable format which is the fastest on mobile devices. Also, calculations in the vertex shaders which normally support the `Tiling and Offset` controls from the material are not included. These have been replaced with per-texture adjustments of position and scale (these controls will still display if you’re using Repeat wrap-mode textures, but they will not do anything).

All shaders utilize the first set of UV coordinates from the mesh. Thus they will work on any of the built-in objects such as the Cube, Plane, Sphere, etc. There is nothing to stop you from providing adjusted UV coordinates such as when working with a sprite sheet or tile map, to pull from only a small portion of the textures, allowing you to keep several objects on the same texture to reduce draw calls.

Individual offset and scale parameters for each texture should allow you to position each background and each Distortion Map where you need it at the size you need. You could for example pull a small sprite from a sprite sheet and then use the full size of a Distortion Map to distort it.



1Background-1Refractor
Shader: Refract2D/1 Background/1 Refractor (Edit...)

Background

|  | Tiling | Offset |
|---|---|---|
| x | 1 | 0 |
| y | 1 | 0 |

|  |  |
|---|---|
| X Offset | 0 |
| Y Offset | 0 |
| X Scale | 1 |
| Y Scale | 1 |
| Refraction | 1 |

Distortion Map

|  | Tiling | Offset |
|---|---|---|
| x | 1 | 0 |
| y | 1 | 0 |

|  |  |
|---|---|
| X Offset | 0 |
| Y Offset | 0 |
| X Scale | 1 |
| Y Scale | 1 |
| Distortion Power | 0.08 |

While it’s ideal for speed that all textures are square and power-of-2, this isn’t required. It is also not necessary to use the same size or format for each texture. Use a large texture, scaled with an offset, to scroll `off-screen` areas into view as desired. Or use a square texture and then scale it to fit inside a rectangle to take advantage of full texture resolution.

**7.1 Refraction**

The Refraction value of each background texture allows you to adjust how the texture will be distorted, independently from how it is set up for each Distortion Map. Whereas the Power adjustment alters the influence of the Distortion Map (positive values apply refraction, negative values apply reflection), the Refraction value of each background is multiplied by the Power of the Distortion Maps. Leaving the Power values alone and adjusting the Refraction values is the easiest way to make adjustments, unless you want to individually alter the contribution from each Distortion Map.

**7.2 Real-Time WYSIWYG Feedback**

Since Refract 2D is shader-based, it renders refractions and reflections in realtime. This means that as you play with the shader controls you can see instantly What-You-See-Is-What-You-Get feedback. If you set up an object in your scene and apply a Refract 2D material to it, adjusting the Material will instantly update the scene view so you can see what the final effect will look like. The Unity Animation Editor window also allows you to build animations of individualy Material properties easily, or alternatively you can animate them from scripts or visual programming or other third-party methods.

**7.3 Limitations**

Most of the shaders in Refract 2D are Shader Model 2, so can run on fairly old hardware with good performance. However, as more backgrounds or Distortion Maps are added, and in some cases including a mid-ground texture, the `Shader Model` runs out of support for the number of inputs and adjustments required. Only when absolutely necessary, Refract 2D switches to using Shader Model 3 in order to support as many textures and inputs as possible. This generally only applies to the more complex shaders.

Additionally, in some of the most complex shaders with many textures, there aren’t enough inputs supported even in Shader Model 3. In this case, certain features of the shaders are omitted in order to support the core functionality. For example you may notice that the third background texture in the most complex shaders does not have its own Refraction value - instead it shares a Refraction value with the second background. Also in the most extreme cases, there simply isn’t enough support for the sheer number of inputs needed to support a mid-ground texture. For example in the 3-backgrounds +

3-refractors shaders, there are no versions with support for a mid-ground simply because Shader Model 3 can't support all the inputs needed for 7 textures.

All shaders currently overwrite pixel colors on the screen. The shader output does not blend with anything existing on the screen, ie it does not alpha-blend with existing imagery. Future iterations of Refract 2D may provide an additional set of shaders to support this. Alternatively you can modify the shaders (see the tips section at the end) to enable blending with existing backgrounds.

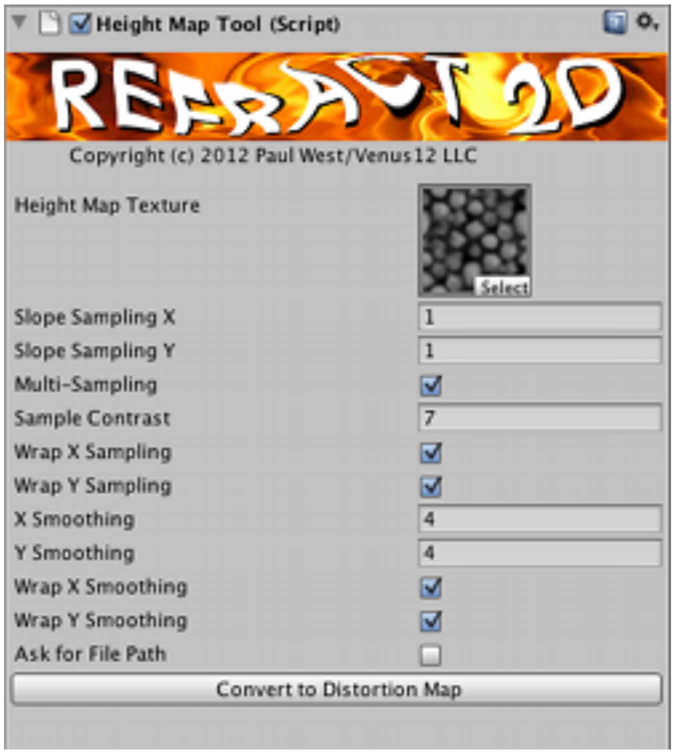
Each shader requires at least 1 background texture in order to be able to distort it, and cannot apply the distortions directly against the backbuffer. This means you need to know what your background `behind the effect` will be before you render it, and this needs to be passed into the shader as a background texture. This could be a static image, for example the image of a wall or background object, or it could be something you rendered to a Render Texture.

For example, if you have a scrolling background with parallax layers, you might render these to the screen first, perhaps rendering the whole screen to a Render Texture, or re-rendering only the portion behind your effect to a Render Texture. This can then pass into the shader and be distorted to create the impression that the background layers are also distorted.

# 8. Height Map Tool

The Height Map Tool included with Refract 2D is a handy utility to convert Height Maps into Distortion Maps and forms a somewhat essential step in the standard workflow. It is set up as a Prefab for ease of use. All you have to do is click on the prefab in your Project window and the user-interface will appear in the inspector. You do not need to add it to your scene.

The tool has a fairly simple interface - just select a Height Map Texture, choose your settings and click the button to `Convert to Distortion Map`. A Distortion Map will be output into your Project folder, either next to the original texture with the word "-DistortionMap" appended to the end,



Copyright (c) 2012 Paul West/Venus12 LLC

| | |
|---|---|
| Height Map Texture | |
| Slope Sampling X | 1 |
| Slope Sampling Y | 1 |
| Multi-Sampling | ✓ |
| Sample Contrast | 7 |
| Wrap X Sampling | ✓ |
| Wrap Y Sampling | ✓ |
| X Smoothing | 4 |
| Y Smoothing | 4 |
| Wrap X Smoothing | ✓ |
| Wrap Y Smoothing | ✓ |
| Ask for File Path | ☐ |

or in a location of your choice with a name you specify. You can cancel the conversion process at any time by clicking the Cancel or (x) button on the progress bar.

## 8.1 Slope Sampling X and Y

The Height Map Tool reads through your Height Map Texture to figure out the slope of the surface depicted in your image. Slope Sampling X and Slope Sampling Y determine how many pixels offset from the current pixel it will read a second sample. This only comes into play when the Multi Sampling option is enabled. Normally you would want to read the pixel 1 pixel to the left or 1 pixel above. However, somewhat smoother results might be generated by sampling the pixel at a greater offset. For example Slope Sampling X of 10 means to sample 10 pixels to the left of the current pixel, while Slope Sampling Y of 20 would mean to sample 20 pixels above the current pixel. When the Multi Sampling option is not enabled, only the current pixel is read and these settings are ignored. Also if Wrap X Sampling and/or Wrap Y Sampling are disabled, sampling of the second pixel will be clamped to the edge of the texture, otherwise they will wrap around the texture edges. Only whole integer values are allowed.

## 8.2 Multi Sampling

The Multi-Sampling option samples the texture at four locations for each pixel, similar to a bilinear filter. The current pixel is sampled, a pixel at Slope Sampling X to the left is sampled, a pixel at Slope Sampling Y above is sampled, and a pixel at offset Slope Sampling X,Slope Sampling Y (diagonal) is sampled. Two horizontal slopes are calculated from this data, along with two vertical slopes. The two sets of measurements are then averaged to produce an average horizontal slope and an average vertical slope. The result is a somewhat smoother bilinear sample of the area. Generally this results in better quality Distortion Maps, but can also be slower to calculate.

## 8.3 Sample Contrast

The Sample Contrast option allows you to specify a floating-point number which adjusts the dynamic range of the Distortion Map. Generally speaking, lower numbers spread out the Distortion Map over a greater range of values, while higher numbers decrease the range. While this may not technically be a `contrast` adjustment, it is somewhat similar. Generally values in the range of 4 to 10 produce a good spread of distortion values from most source textures. If your image seems to only produce a narrow range of values or produces a Distortion Map that looks faint or mostly yellow, you may want to decrease the contrast by lowering this number. Ideally you want to make full use of the dynamic range of pixel values to represent your surface with the most precision. Distortion Maps with too much contrast may result in blockiness or graininess in the final effects, while too little contrast may result in shallow bumps which become much uglier/blockier when the Refraction value is increased.

### 8.4 Wrap X and Y Sampling

When Multi Sampling is enabled and the tool is reading samples offset from the current pixel by 1 or more, it's possible that it may need to read pixels off the top of the texture, or off the left edge of the texture. These adjustments allow you to tell the tool whether to let sampling wrap around the edges of the texture (to sample pixels from the opposite edge if needed). If disabled, sampling will be strictly clamped to within the texture bounds. If you are seeking to produce a seamless Distortion Map which can be scrolled and wrap around without a join, you may need to enable the wrapping options. An alternative approach is to make your Distortion Map texture seamless after it has been generated, using some other graphics tool. If you don't want your texture to wrap, you may not want pixels to `bleed` in from opposite edges of the screen, and disabling the X and Y sampling allows this to be managed.

### 8.5 X and Y Smoothing

Height Map Tool includes the ability to apply a gaussian blur  to the Distortion Map after it has been generated. The X Smoothing and Y Smoothing adjustments tell the tool over how many pixels to sample/blur the texture. A good starting point is a value of 4, which means that values from 4 pixels will be averaged to produce a blur. The purpose of blurring a Distortion Map is to smooth out the finely-detailed `grain` when it is not desired.

There are times when you want no blur at all, such as for frosted glass where you want lots of tiny details - here, blurring would lose all of the details. Smoothing can be disabled individually in the X or Y directions by providing a value of 0. Therefore you can do a horizontal blur but not a vertical blur, or vice versa. When both directions are blurred it simulates a gaussian blur centered on the current pixel.

When you are seeking a particularly smooth, clean distortion, you will want to blur the Distortion Map. Even if you blur the Height Map texture, you will find that tiny relative differences between individual pixels are enough to produce bands or aliasing artifacts in the Distortion Map, because the tool interprets the change in pixel values to mean there is a `slope`. To produce a really smooth distortion which looks antialiased you will need your Distortion Map to be heavily smoothed - up to as much as 50-100 pixels of blur size. While the Height Map Tool can smooth for you, you will probably find that the smoothing step is relatively slow and may not be of as high quality as you've seen in graphics software. Except for matters of convenience, it may be better to bring the Distortion Map texture into your graphics software to clean it up, smooth it, make it seamless, or make other adjustments to it, then save it back to your project.

## 8.6 Wrap X and Y Smoothing

These options tell the tool whether, when performing X and/or Y smoothing with the previous options, to allow pixels to be sampled off the edges of the texture. If enabled, the texture will be sampled from the opposite edges of the texture if there aren't enough pixels within the texture bounds to get a complete set of samples. If disabled, smoothing will clamp to within the bounds of the texture, which may produce slightly less blurry results at the edges of the texture. If you are using the smoothing and you want your Distoriton Map to be scrollable/wrappable, then it's a good idea to enable the wrapping of the smoothing. Still, faster and better quality smoothing can be achieved in dedicated graphics software which you may prefer.

## 8.7 Ask For File Path

The Ask For File Path option if chosen simply shows a file requester window where you can choose the output location of the Distortion Map and its name. Otherwise a texture is output with the same name plus "-Distortion Map", in the same location.

## 8.8 Acceptable Height Map Textures

Height Map Tool can accept Height Map textures of any size and proportion. They must be in either Alpha8, RGB24, RGBA32 or ARGB32 format. The tool is hand-written to work with these texture formats only and will show a message if you try to use other texture formats. The easiest way to specify a valid format is to mark the texture as `Advanced` in the import settings and choose the desired format. The tool cannot read compressed textures or any other formats. The texture must also be `Read/Write Enabled`, since the tool must be able to read the pixels from the texture. If it is not, the tool will prompt you and offer to make the change for you. Bear in mind that a texture that is Read/Write enabled will store a copy of the texture data in main memory as well as in graphics texture memory.

## 8.9 Alpha Channel

Your Height Map texture does not need an alpha channel (unless it is an Alpha8 texture), but you can optionally include one. When your texture is RGBA32 or ARGB32 format, the alpha channel will be transferred into the final Distortion Map. In this case, the Distortion Map texture will be generated in RGBA32 format. Otherwise the Distortion Map texture will be generated in RGB24 format. Only the Red and Green channels are used for distortion, the Blue channel is currently reserved for the future. Even though the tool will preserve your alpha channel, Refract 2D does not currently make use of the alpha channel of Distortion Maps - this is also reserved for the future.

**8.10 Texture Resolution**

You might be surprised how low-resolution your Height Map and Distortion Map can be and still look smooth. In fact, in some cases a larger texture with more detail will produce a grainier effect than a very low-resolution texture. For example a 64x64 Distortion Map will be bilinearly interpolated (if you choose this in its Import Settings) and this effectively produces a blurring of the data, resulting in smoothness. Highly smooth distortions don't need to be extremely high resolution textures, unless you particularly need the precision.

Additionally, you will find that textures you distort using Refract 2D don't necessarily need to be high resolution either. In the case where there is no distortion, indeed you would prefer one texel for each pixel on the screen so that your image is crisp and detailed. However, once your image is being distorted, it is possible that any given pixel or lens in the Distortion Map has access to sampling *any* pixel from your texture. The texture is effectively resampled, producing a potentially much higher resolution image as the result. Coupled with bilinear filtering this can produce extremely smooth and high-resolution-seeming results even from a relatively low-resolution image texture. So don't be caught in thinking you need to use huge textures in order to get better quality results - and of course lower-resolution textures can save on memory and increase rendering speed.

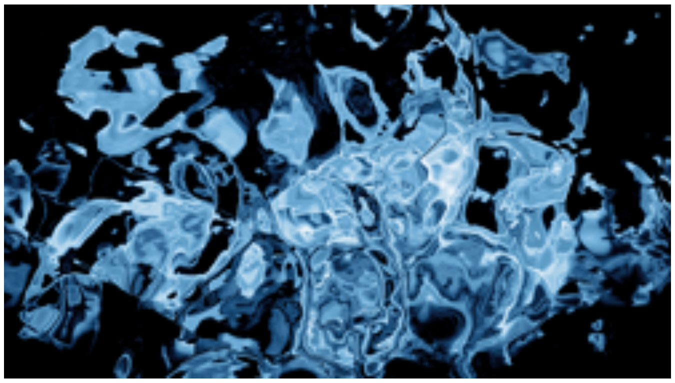
# 9. Example Scenes

Refract 2D comes complete with a collection of example Unity scenes, set up and animated ready to demonstrate various possible applications of the Refract 2D system. Refractions, reflections and light mapping provide many creative options, so encapsulating them all in set of scenes is not possible. Hopefully the scenes will inspire you as to what might be possible using Refract 2D in your games or applications.

## 9.1 Scene 1 - Fire

The Fire scene shows how it may be possible to animate realistic-looking flames using refractive lenses. The scene uses a single background - an existing image of a real-life flame - and *four* Distortion Maps. The Distortion Maps do all the work as they scroll up the screen at various speeds. As they move, their distorting qualities combine with each other to produce an overall distortion of the background. This dynamically distorts the image of the flame and results in somewhat realistic fire.

## 9.2 Scene 2 - Ice Flame



The Ice Flame is a nice visual effect combining the idea of real-time fire animation with, well, some ice cubes. One of the Distortion Maps remains stationary, comprising the contours of some blocks of ice. With the blue coloring, this gives the flame something of an icy feel. A single background image of a hue-shifted real-life flame is used, which is then distorted in realtime from three combined Distortion Maps. The final result is cool, refreshing effect.

## 9.3 Scene 3 - Stained Glass

The Stained Glass demo portrays a beautiful stained-glass window. The same original image of a stained-glass window was used to generate the actual Distortion Map as to color it. This was then blurred. The Distortion Map was then coupled with the original colored image as a `mid ground` texture. The color image is not distorted by the Distortion Map, but a background Light Map and a background Environment Map are distorted, to give the impression of movement behind the scene. As the window scrolls and the background moves, color from the mid-ground texture colorizes the light coming through the window, showing off the smoothly contoured stained glass.

### 9.4 Scene 4 - Liquid Metal

The Liquid Metal scene demonstrates realtime bumpy liquid animation in a 3D environment. Although Refract 2D is technically a `flat` system of distortion that doesn't take perspective or rotation into account, it can still be used quite effectively in various 3D scenarios. Here, dual Distortion Maps move and combine to create a dynamic liquid surface, colored by a background texture and lit by a light map texture. Can you tell it's not perfectly 3D?

### 9.5 Scene 5 - Plasma

The Plasma scene puts Distortion Maps to artistic use, combining 3 background textures and 3 Distortion Maps with an abnormally high degree of Refraction. This produces a beautifully artistic plasma effect. To an observer, you probably can't tell that this is based on lens refraction. The high degree of Refraction pulls pixel colors from a wide range across and outside of the texture bounds, compacting the colored bands into narrow undulating rings of color. Since this is a very input-heavy shader, Shader Model 3 is used, plus the Refraction value of the second background has to be shared with the third, hence there are only 2 Refraction controls.

### 9.6 Scene 6 - Heat Wave

Experience the heat of the desert in this fun heat-wave simulator. Watch those camel-riders wiggle as the shimmering, scorching heat rises up. Is it starting to feel hot in here? The effect is fairly simple - sometimes subtle distortions are useful. A single colored background is distorted by two simple Distortion Maps. With a very low degree of Refraction, the result is a slight wiggle, just enough to give the impression of heat distortion on a hot day.

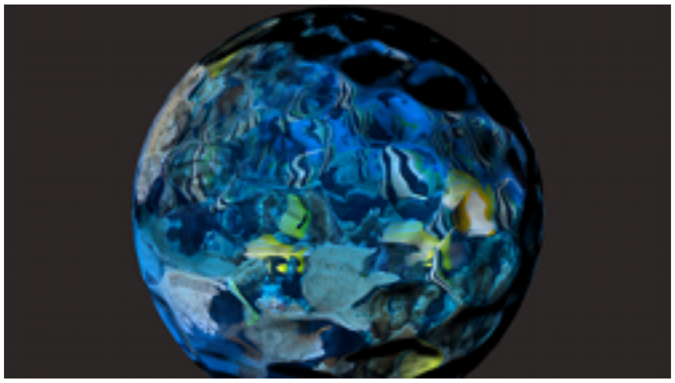
**9.7 Scene 7 - Frosted Glass**

Sometimes you want highly detailed refractions or reflections, perhaps representing an organic earthy environment, a metal or wood surface, or perhaps a frosted window. The Frosted Glass demo shows the kind of Distortion Map that can be produced when the Smoothing functionality is switched off in the Height Map Tool. By providing a value of 0 for X Smoothing and Y Smoothing, and starting out with a crisp in-focus texture, tiny refractive or reflective details emerge to create that high-detail or frosted look. This demo transitions between colorless glass, which frosts the background, to colored glass, which combines frosting with a glass tint.

**9.8 Scene 8 - Distortion**

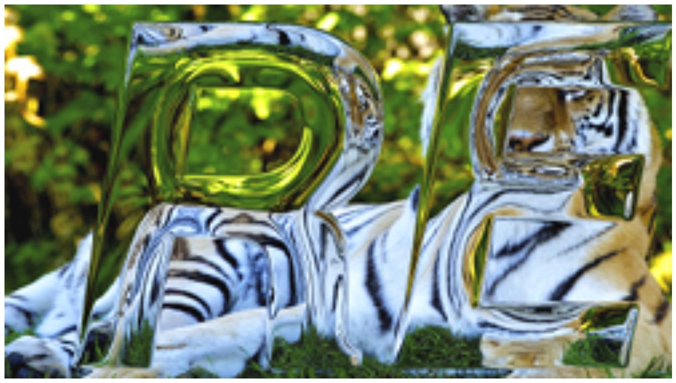
Okay, so all of the scenes show distortion, but the Distortion scene especially shows how you can transition between different Distortion Maps, transitioning between refractive and reflective effects in realtime. Four Distortion Maps are used, yet at times any of those Maps may be active to varying degrees. The Power value of each is animated to transition from refractive strength to reflective strength, and at times returns to a Power of 0 to disable effects from a given Distortion Map. Sometimes more than one, or up to four Distortion Maps are in effect. You can see how `holding` distortion from one Distortion Map while adding-in distortion from another, dynamically modifies the distortion effect. The distortion itself is distorted!

**9.9 Scene 9 - Aquarium**



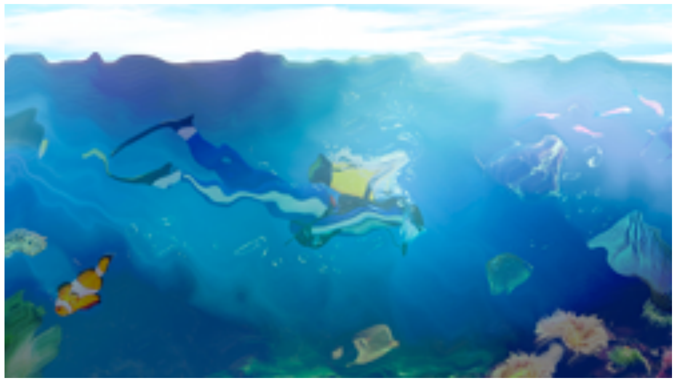
Watch the fish swim in the spherical aquarium. How they got inside, nobody knows. This demo shows an exaggerated Light Map. Refraction of the Light Map is increased three-fold so increase the intensity of the distortion. This creates an increase in the contrast produced by the light shape. Notice in the lightmap that some areas are light and some are dark - the light areas obviously represent light while the dark areas represent shadow. The over-refracted Light Map produces deep bumps on the aquarium surface, which animate between refraction and reflection in realtime. A subtle environment map is also used.

### 9.10 Scene 10 - Text



Sometimes when you've made a nice logo or show a text message, you want it to reflect an environment, perhaps react to a light map, or distort a background so as to appear like glass. Here, the Text demo shows a large rectangular Distortion Map used in conjunction with a square background texture. Offset and Scale are used to correctly size and animate the text across the colorful background. A high degree of Refraction is used to make the text appear to be highly refractive like a high-powered lens. The text was generated first in a graphics application, then a copy of the layer was made. The copy was blurred heavily and then multiplied by the original text to keep a hard outline edge. This resulted in a smooth interior but defined edges, and coupled with the high degree of Refraction results in beautifully smooth glass text. This is achieved with a single Distortion Map generated from the text Height Map, and a single colored background. It is possible with the right Height Map to give text any kind of contour, bevel, or surface shape.

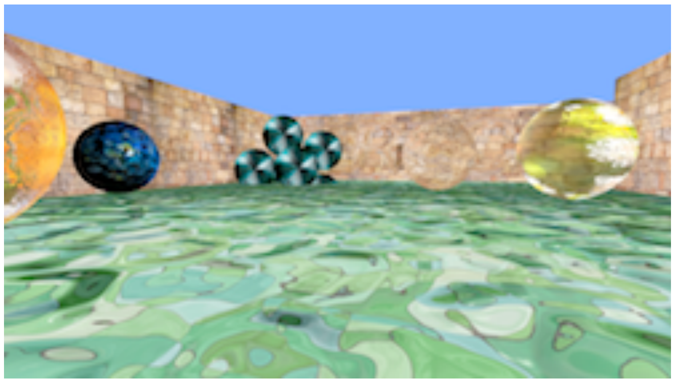
### 9.11 Scene 11 - Water



In the Water demo, special techniques are used to animate a 2D water surface, complete with underwater distortions, lighting, and an un-distorted sky texture, all within a single shader. Two Distortion Maps animate the water ripples and surface. Care has to be taken to avoid over-refractive effects since this would result in blobs of water appearing out of thin air. The water surface ripples and animates with an interesting effect, yet does not distort the sky. While this isn't a complete simulation of interactive water, movement and positioning of distortions near the surface can produce pleasantly rippling water effects.

The effect is achieved through the use of an alpha channel. The first background includes an alpha channel, with values of 1.0 (or 255) where the water/diver scene is to be shown, and a 0 where sky is to be shown. Provided this texture is used as the first background, and the texture is RGBA32 or ARGB32, the alpha channel can be used by the shader. The light map is distorted by the same two Distortion Maps and, provided the Refraction value is the same for both textures, the water/diver and the lighting will both refract in unison, creating a seamlessly lit water scene. The final touch entails the use of the light map as the third texture, including a strip of white pixels where the sky needs to appear. This is because the values from the light map will be multiplied with the scene - we want the sky to show its own colors so we allow it to be `multiplied by white` in order to retain its full color. The finishing touch is to move the clouds by individually scrolling the sky texture. With some care, realtime caustics effects could be produced on the underwater objects as well. A wide variety of watery effects could be accomplished with Refract 2D in a variety of ways, and this scene demonstrates just one of them. Have fun experimenting to find the exact-right effect for your game or app.

## 9.12 Scene 12 - 3D



The 3D scene demonstrates most of Refract 2D’s other demos all in one interactive environment. You can move around with the arrow keys on your keyboard and move the view with your mouse. With this demo it is possible to view various effects in 3D, get up close, or at a distance, or see them from acute angles to grasp the ways that they do or don’t work the way you might expect in a 3D world. You should find that rendering performance with this demo is very fast even if the entire screen is covered with distortions - some other reflective/refractive methods can be much slower.

What you can see here is that, depending on your needs, sometimes Refract 2D works well in 3D, and in other situations it doesn't. For example, I personally think the green water effect covering the ground looks really nice - sure you can't go under the water, it doesn't really have depth, it's not real `3D`, and the lighting may not be technically correct, but the rendering quality is good. If this fits your 3D game in certain situations, then that's great. The water also shows that you can apply lighting with a light map and make it look reasonably realistic.

The spheres demonstrate distortion of a wall texture, frosted glass, the aquarium demo and the liquid metal demo. You'll notice that the balls situated near the corners of the arena close to the walls are actually passable as refracting the wall behind them - this illusion seems to make sense so long as you don't get too close - once the balls start to overlap the sky or the water without refracting them, the illusion is broken. Ideally the background behind the spheres should be captured to a RenderTexture, in screen-space, and then have that grab be warped to fit the world-space of the object, but in this demo I haven't tried such measures.

Also on the right wall you'll notice a heat distortion, it looks reasonable given it's flush against the wall because the flatness of its surface corresponds to its proximity to the flat surface behind it. But to the left of the arena is another instance of the heat distortion, standing out as what looks like a separate wall - really this just demonstrates that the distortion is flat and warps the immediate background texture, which here is distant from the real wall of the arena, so instead you see what looks like a solid wall with heat on it instead of a refraction of the arena walls behind it. It should really look transparent and not like a solid wall.

The text demo scrolls across the near wall (turn around to see it), and looks nice from a distance, but up-close or at acute angles you realize it is totally flat, even though it looks 3D when standing back. Again if this works for you in your situation that's great, but if your player is going to get up-close and personal it may break the illusion. Refract 2D’s distortions are applied on a flat surface and create an illusion of 3D distortion, when viewed straight-on, but other angles to the camera are not taken into account.

The liquid metal demo (the set of moving balls) features a moving light source, as a light map, but without this corresponding to some other light source in the scene it doesn't totally make sense. Indeed the balls turn dark at some points and this doesn't seem to match the scene lighting. In some situations this would be fine, and you could fudge it to make it seem as though the objects are lit in a way that they would be lit if reflecting a real 3D light source, but here the illusion is a little broken. Again in certain circumstances this would be acceptable, but not in others - it totally depends on your environment, when and where you use the effect, and how you play to the effect's strengths and hide its weaknesses.

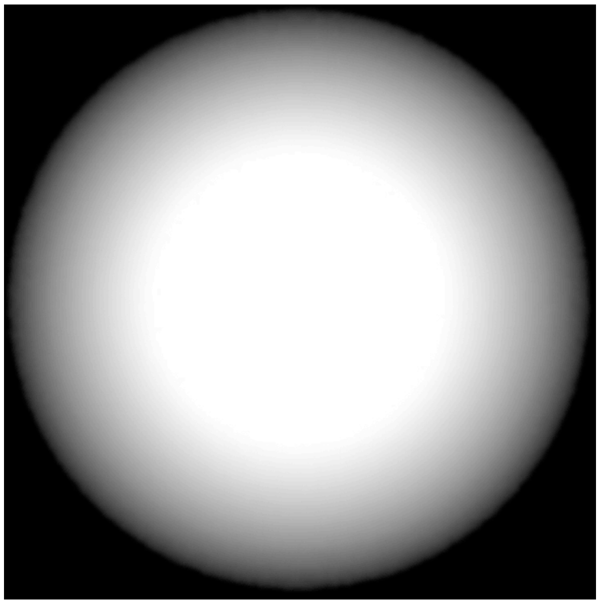
With some careful choices, there are many many things you could possibly use Refract D for in a 3D game. For example it might be appropriate to simply distort a 2D image for some reason. Or maybe you want a realtime fire effect but it's not something the player can get up close to. Or maybe you have Unity Pro and can play around with RenderTextures to create dynamic distortion maps or perspective/rotation-corrected refractions, etc.

Hopefully this demo shows you how Refract 2D will behave in a 3D game - it has its uses, and some of them are quite fascinating and cool, but it also has its limits. Mainly you’ll see that Refract 2D works on a flat surface and creates the illusion of refracted and reflected light, but is designed to only look correct when viewed straight-on. At other angles the light reflections are incorrect. From afar the effect can be convincing enough that you can’t really tell, and maybe don’t care, that it isn’t perfect. But often when up-close you’ll start to experience some of these issues in a more obvious way. It’s up to you whether you use Refract 2D in a 3D environment, and hopefully you can put it to good use.

# 10. Lighting

**10.1 Light Maps**

Refract 2D uses a `light map` texture to implement lighting. Light is stored in the texture, in any color or shape, and is multiplied by the rest of the output. A light map is effectively a cross section of the light being emitted near to the surface, sampled on a per-pixel basis. So a slice is taken cutting through the light parallel to the surface, and is stored in the texture. It is similar to a single slice of a brain scan which shows a 2D view of a slice of the brain, except we're taking a 2D view of a slice of volumetric light. This shows a cross-section of the light, and therefore also the lack of light, which is shade. Of course white is bright light while black is no light, and lights can be any color on a per-pixel basis, which means you can have as many visible lights as there are pixels!



Since the light map is just a texture, you can store absolutely any image in it and use that as `light`. This could be a panorama of a scene, providing image-based lighting. Perhaps this image might be highly blurred to soften the general surrounding light. It could be something you rendered, an image of an environment (as in environment mapping), a collection of realtime generated light sources, some other pre-drawn image, an animated `light sprite`, `light tiles`, or perhaps something being emitted by nearby surfaces.

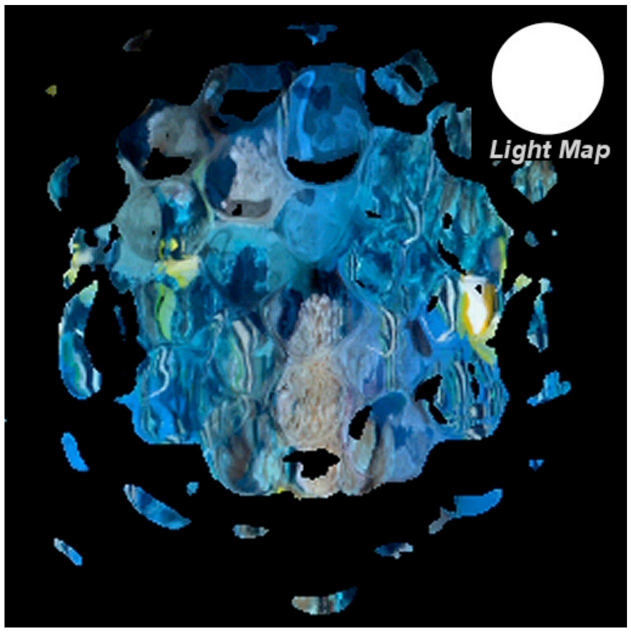
Although in 3D digital environments you might have become accustomed to thinking of light as something different from a texture, e.g. a realtime-generated point light based on a math function, in the real world when a surface is at all shiny (ie reflects or emits some light and doesn't absorb it all) it simply reflects what's in the surrounding environment. So real-world lighting really is the same thing as environment mapping, which is basically reflection + emission. When you see what people refer to as a `specular highlight` for example, it merely means that somewhere in the surrounding environment there is a small, limited-sized light source like a light bulb, and it shows up in the surface of the object as a small dot. Often this bright spot stands out from the rest of the reflections as a highlight, which is why they're called specular highlights. But in addition to the specular highlights are all kinds of subtler and perhaps less obvious or dimmer reflections of light coming from other surrounding objects, even if those objects are only passing on light that they received from somewhere else. So ultimately, lighting is just environmental reflections and ideally every object in your environment should reflect its environment. This is why cube-mapping and image-based-lighting for example are popular as ways to model environmental light in 3D.

So in Refract 2D, mathematical lights that are limited to approximating diffuse and specular light from point lights are thrown out, to be replaced with per-pixel light information stored in a light map texture. This provides far more control over the color and position and intensity of light across the whole screen on a per-pixel basis. To do something similar in 3D would require massive volumetric 3D textures. 2D rocks!

## 10.2 Hard Lights

A single white pixel in the light map represents a tiny point of light in the environment. Or to put it another way, that single white pixel could be a reflection of a small LED light bulb in an otherwise black room, for example. When you think of your Distortion Map surface as shiny and reflective, and you think of lighting as simply environmental reflections, the Distortion Map basically reflects the light map texture, using it as a picture of the environmental light.

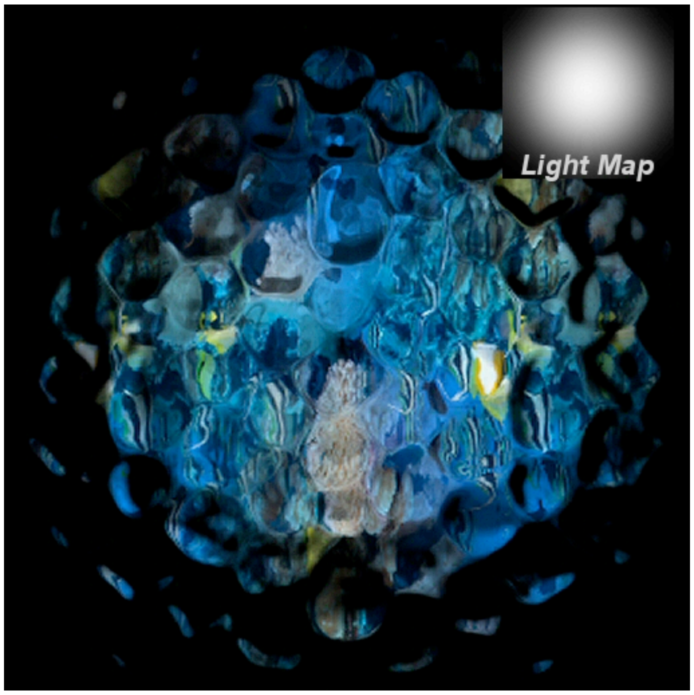
So a single pixel in the light map will be reflected by the bumpy surface and show up as a tiny dot, seeming to light the surface in one small spot but nowhere else. All other pixels will be output as black because all other pixels in the light map would be black. As the light source/texture moves, or as the surface moves, that dot of light will dance around according to the surface distortion/bumps. You might think of this one-pixel light as showing up in a similar way to specular highlights. It might represent the brightest point at the center of a light source. It also suggests that the light source represented by that one pixel is `one pixel wide`.



As we increase the size of our light source, we could make it a larger circle of white pixels. At the edge the circle immediately cuts to black with a hard edge. This could perhaps represent a spot light, a flashlight, or a ball of light which is equally bright all over. This will be reflected by the surface, but you will then notice that the hard edges of the circle show up as hard edges of reflection. The surface will simply reflect what it sees in the light map. You may or may not want such a hard-edged effect - it's almost like a very large specular highlight from a uniformly outputting light source.

## 10.3 Diffuse Lights

Sometimes you will want to have very crisp reflections like this, since you might have an image of an environment in your light map texture and want to see the fine details. Often though when you think of lighting you often think of diffuse light, where the rays of light might have spread out as a result of bumps in the surface of the light source or environment, or perhaps where diffuse light has been created by being bounced off other objects. For example radiosity or ambient occlusion may have occurred in some areas. You might also want your light to be generally softer. In this case, all that is required is to blur the light map. This softens the light and disperses it. Basically what a blur does is, takes the light from each individual pixel and spreads it out to the surrounding pixels. So this `diffuses` the light. Blurring the whole light map fairly heavily provides what will look to be diffuse lighting.



What you probably want to achieve in most cases is something resembling point lights. A point light basically has some kind of central source and then light radiates from it and diminishes over a distance. At some distance the light becomes so dim that it no longer lights objects, having been scattered by collisions with air particles, dust or fog for example. This can be achieved by putting a local light source into the light map - ie, basically our one pixel, or our filled circle, or a blurred circle, or some other shape that isn't the same across the entire light map. All these shapes, indeed any shape, can be used as `local` light sources. They don't necessarily have to be `points` because we can manage the shape of the light source on a per-pixel basis. So you could have square lights, ring lights, lighted wavy lines, randomly scattered point lights, oval lights, laser beam lights, triangular lights, light shafts, etc. If you can draw it to a texture, you can use it as light sources. Again, 2D rocks!

## 10.4 Ambient Light

Ambient light can be achieved simply by making sure that all of the pixels in the light map texture have at least some brightness to them. Indeed you can control the ambient light on a per-pixel basis, plus the ambient light can be any color on a per-pixel basis. Ambient light, point lights, cone lines, whatever-shape-lights can all be blended on top of each other into a single light map. Even an image of a household object could be a light source - your cat or dog, or your best friend's face. Whatever you want. This is why image-based lighting is so powerful and versatile - it simply goes back to the fact that lighting is all about either direct emission of light or reflection of emitted light off of other objects.

### 10.5 Specular Highlights

You can combine types of lights in a light map, given it is pixel-based, using whatever graphics software you have at hand. For example you can create diffuse light by blurring your light sources, then overlay the original crisp un-blurred light sources on top. This gives you diffuse + specular lighting as shown here.

Specular light is simply highly concentrated `sharp` light while diffuse light is simply `blurred` light. There's nothing to stop you from having any number of levels of specularity/ sharpness in the same light map - if you can draw it you can use it. Perhaps then add-in some ambient light and whatever else you want to do to it and you're good to go. You could even get really advanced and use some super-sophisticated lighting model. Or perhaps take a photo of a panorama and blur it to utilize beautifully natural environmental light. And don't forget that the light map is an RGB color texture, so you can use multi-colored lights as much as you like!

Light can also be animated. After all, it's just a texture. Using RenderTextures for example you can render realtime light animations to it and watch those lights move in realtime. A video sequence could even be a light map, or a realtime-generated effect, or a sequence of animation frames. It is limited only by your imagination.

### 10.6 Dynamic Lights

A useful technique in a 2D game environment is to have a light map that is about 2x the width and height of the screen. Bumps near the edge of the screen that point outward, as if reflecting light from outside of the edges of the screen, need to still refer to light sources that are `off-screen`. So by maintaining a larger light map texture centered over the scene you can accommodate this without seeing wrap-around or clipped light reflections. In fact, because Refract 2D resamples textures based on the surface and this can act to increase the effective resolution of the texture in realtime, you can actually get away with a much smaller light map texture than you'd think you might need. Especially if most of your lighting is fairly diffuse ie blurred, there is little need to store it in a high resolution texture. Smooth, highly-blurred lights for example could be stored in a significantly smaller texture, and bilinear or trilinear filtering on the light map texture's import settings will deal with smoothly interpolating the light for you.

With this in mind, it is possible to store the light for a fairly large game environment in a single large texture. By scaling the light map significantly and relying on smooth interpolation, the light map can remain static and cover a large scrollable area covering many screens. This is one way to compensate for the absence of RenderTextures in Unity Free, allowing you to light a good-sized 2d environment without having to re-generate the light map on the fly. However, once you do gain access to RenderTextures there are many interesting possibilities that open up including animation, higher-resolution scrolling light maps, and the ability to only render the visible lights to the light map each frame.

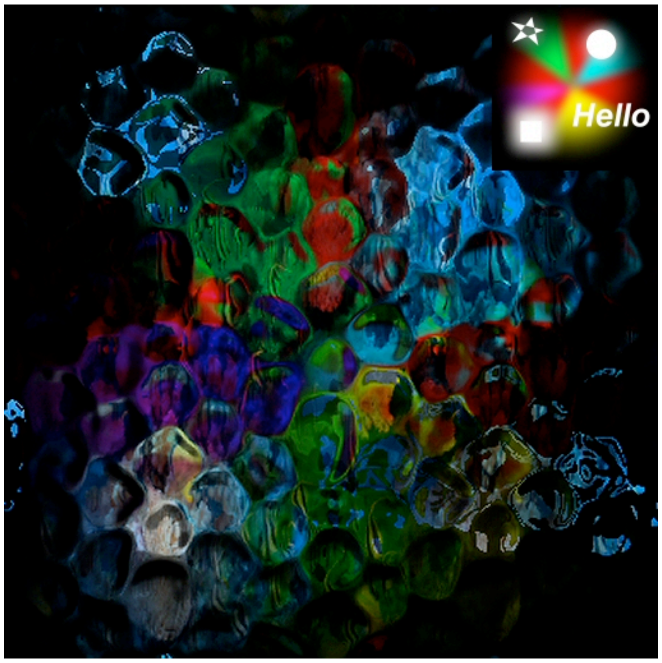
## 10.7 Shadows

Refract 2D does feature some `shadowing` ability. It is not a complete, or totally accurate shadowing solution, but it can prove useful. The darker areas of a light map effectively tell the Distortion Map where there is `no light`, or less light. In the case of a hard-edged circle for example, the areas outside the circle are not emitting any light so are dark. You could say that the dark areas are shadows. When you apply this light map to a bumpy surface you will see that pixels closer to the circle receive white light, while reflections that land outside the circle receive black pixels. For example on the inside of a crater or `pit` in the surface, the edge pointing away from the circle will likely pick up black pixels, while pixels pointing more towards the circle will pick up white pixels. This creates the impression that the edges of the surface that are `pointing away from light sources` receive shade, while the surface that `points toward light sources` receives light. This can create a fairly convincing illusion that the surface is rendering shadows, based on the simple fact that the light map texture has some areas that are lit and some that are shaded.

The surface is not technically self-shadowing. It does not pay attention to whether a given slope can actually see the light source pixel it points at, unobstructed by other bumps. This is one flaw in the realism. However, you will find that, similar to the sun, the further away from the center of a light source the surface pixel gets the `longer` the `shadows` get. This is because the angle to the light is becoming flatter and requires light sources to be much further away in order to reflect them. Just as the late afternoon/evening sun casts long shadows, so too will it seem, to a degree, that further-away bumps in the surface will `shadow` more, ie will not be able to reflect the light.

So for a `point light` in the light map texture, it will behave somewhat like Unity's point lights where surfaces further from the center of the light are darker. It will seem that the light source is a short distance above the surface rather than completely flush to the surface, which is why a) this does not cast lines of visibility that totally obscure `bumps behind bumps`, and b) `bumps behind bumps` may still be lit (albeit less) because they still point to enough of the light source to receive some light. Typically with more blurred/ diffuse light maps this is more noticeable, because the larger the area covered by the light in the light map the further it will `reach` to remove bumps.

Using soft blurred diffuse lights generally creates what looks like fairly realistic `light and shade`, but does not cast shadows from one bump onto another. So a Distortion Map surface provide some impression of  `shadowing`, but not `self-shadowing`. Self-shadowing would be required in order to cast obstructive shadows.



So you can emulate pretty much any kind of light in Refract 2D and being able to modify the light on a per-pixel basis is a great boost for creative options, subtle quality and control.

Note that decreasing the Refraction value on the Refract 2D shaders into a negative number converts it from refractive to reflective. `Over-reflecting` a light map artificially warps the surface angles creating the illusion of much more prominent contours. For example with `-3 Refraction`, this will bring out even more exaggerated light and shadow which can help to make the surface look especially bumpy.

# 11. Final Tips

A whole host of effects can be produced using Distortion Maps, either with one or several in combination. Refractions, reflections, lighting, bump mapping, realtime image warping, special effects, cool animations, transitions, and more. Since the system is fairly open and flexible your own imagination truly is the limit of what can be achieved.

Here are a few final points.

## 11.1 Removing Texture Edge Bleeding

You will find that if you distort an image, sometimes pixels from the opposite edge of the image wrap around into view, mainly if your background texture is set to Repeat wrap mode. This can be undesirable - you probably don't want to see the hard edges of your texture if it hasn't been prepared for wrapping. Either you can make it seamless, or you can adjust the X Scale and Y Scale of the background in the material. For example, setting it to 0.9 instead of 1.0 makes the texture slightly bigger than the scene, hopefully ensuring (unless you have really strong refraction or reflection) that the edges of the background are never exposed. This is used extensively in the demo scenes to hide the undesired appearance of texture edges at the sides of the screen. If you want to use stronger refraction or reflection you may need to use textures that are larger than the area you area rendering so that you can correctly sample `off-screen` pixels without wrapping.

## 11.3 Modifying the Shaders

Feel free to modify the shaders to suit your needs. Currently, where possible, `fixed` data types are used since, on mobile devices they are the fastest. These are fairly low-precision but have enough to represent colors. If you find that you're seeing too much inaccuracy in the surface or it seems like the effect is too blocky or has `steps` in it, then you might want to increase the data types to `half` or `float`. Simply doing a find-replace operation on the shader sourcecode to change all occurrences of the word `fixed` into either `half` or `float` should work.

You'll also notice that at present, the shaders output solid color, ie whatever is drawn overwrites any existing background completely. This is because `blending` is switched off. You can easily enable blending by changing the line `Blend Off` in the shader sourcecode to something valid such as `Blend SrcAlpha OneMinusSrcAlpha` which will implement alpha-blending. However, varying alpha values are not output from the fragment shaders at this time. You would need to modify the shader to output alpha values from some texture or variable source in order for them to be used with alpha blending or other blend modes. However, some blends that you can achieve without alpha values include Additive, Soft Additive, Multiply, Multiply 2x and Screen. Also using the BlendOp you can do blends like Min, Max, Subtract, etc, without having to modify the fragment shaders.

In all shaders, back-face culling is switched off, since this is mostly a 2D system and in 2D there are generally only front-facing surfaces. However, in 3D applications you might want to change it to `Cull Back` to remove back-facing surfaces. Also Z-buffer writing is switched on and Z testing is set to Less-than-or-Equal (LEqual), meaning you can still use Z-testing in 2D or use the Z coordinate for layering. If you don't need it, using `ZTest Always` and `ZTest Off` may give you some performance benefits. Also it is possible to remove some of the inputs you don't need (if you also remove the code that uses them) in order to improve performance in special circumstances where you don't need control.

## 11.3 Thank You!

Thank You for supporting our development of Refract 2D and other projects. If you have any questions, feel free to ask on the Unity forums or contact the author here:

http://forum.unity3d.com/members/17044-imaginaryhuman

I look forward to seeing your creative uses of the Refract 2D system in your games and applications!