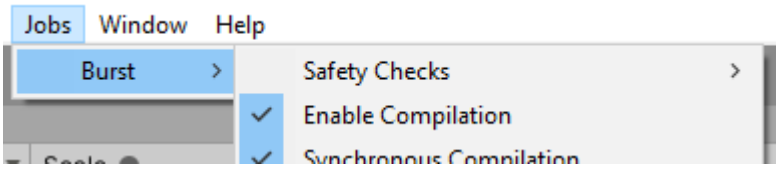
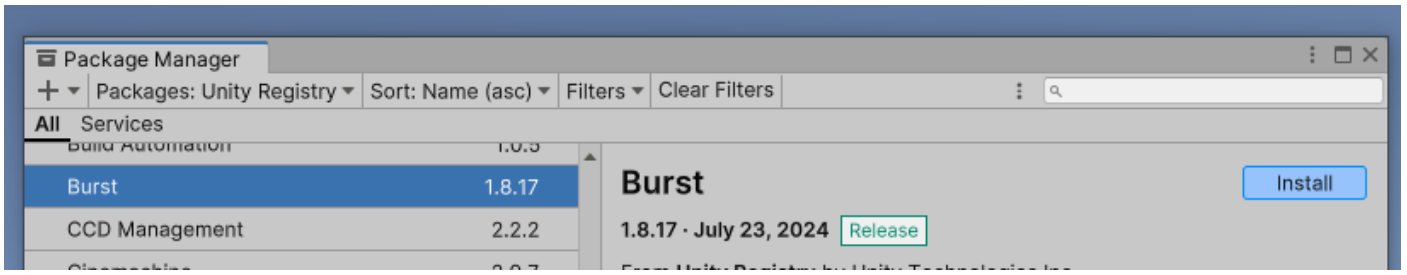
Planet Forge

Documentation

Dependencies

Before using Planet Forge, you must install the **Burst** package from the **Window / Package Manager** inside Unity. Once installed, make sure it's enabled for maximum performance.





Getting Started

The planet example scenes are in the **Plugins/CW/Planet Forge/Scenes** folder. These show you what a fully configured planet with atmosphere looks like with different settings.

The tutorial scenes for the planet surface are in the **Plugins/CW/SpaceGraphicsToolkit/Features/Landscape/Examples** folder. These teach you about the major components and settings for the planet surface.

The tutorial scenes for the planet atmosphere are in the **Plugins/CW/SpaceGraphicsToolkit/Features/Sky/Examples** folder. These teach you about the major components and settings for the planet atmosphere.

Make Your Own Planet

To make your own planet, I recommend you begin with one of the example scenes in the **Plugins/CW/Planet Forge/Scenes** folder. For example:

In the "**Alythar**" demo scene, you can find the "**Alythar**" GameObject.

This GameObject has two children: **SgtSphereLandscape**, and **SgtSky**.

The **SgtSphereLandscape** GameObject has the **SgtSphereLandscape** component. This component has the **Radius** setting, which you can set to your desired value.

If you select the **SgtSky** GameObject, you can see it has the **SgtSky** component, with the **InnerMeshRadius** setting. Make sure this setting matches or is close to your planet surface **Radius** value.

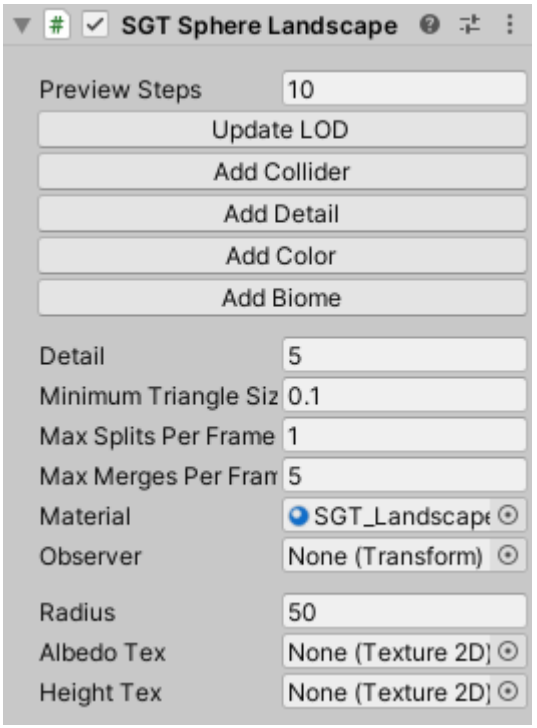
On the same **SgtSky** GameObject, you can click the **SgtSkyDepthTex** component's **Randomize Hue** button to quickly change the color. You can also see the **SgtSkyLightingTex** component, and click the **Randomize Sharpness RGB** button to change the color of the night/dark transition.

Next, you can see the **SgtSphereLandscape** GameObject has a child called **SgtLandscapeBiome**. If you've changed the **SgtSphereLandscape** component's **Radius** value, then you may want to increase the **SurfaceScale**. Next, you can click the various **Randomize** buttons to see what different settings look like.

If your planet is very large, you may want to add an additional layer to the **Layers** list. If you add an additional layer, I recommend you make the **Size** value smaller than the size of the previous layer. For example, if layer 0 has a size of 100, then layer 1 can have a size of something like 14. I recommend dividing the previous size by a prime number, so the tiling is less apparent.

SgtSphereLandscape

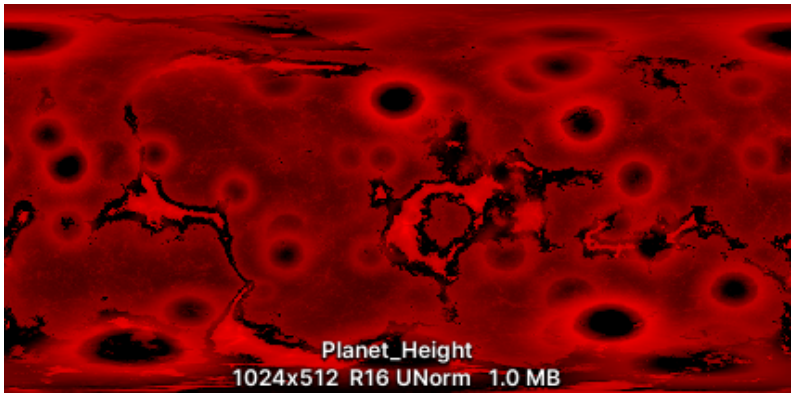
All planets begin with the **SgtSphereLandscape** component, which you can add to a new GameObject. This component will generate a sphere with the specified **Radius**, and the mesh detail will automatically increase as the **Observer** (default = MainCamera) approaches. The mesh detail will increase until the triangles are near the **MinimumTriangleSize** value, which you can adjust depending on your project detail and performance requirements. The **Detail** setting adjusts the overall detail of the planet based on the observer distance.



If you want the planet to generate faster, then you can increase the **MaxSplitsPerFrame** setting. Keep in mind this has a performance penalty, so you should experiment with different values depending on your project.

If you have a pre-generated **AlbedoTex** and/or **HeightTex**, then you can set them. Keep in mind these must use cylindrical (equirectangular) projection. The HeightTex must be **Single Channel** with **Read/Write** enabled.

By default the planet in the editor will update **PreviewSteps** amount of times. If you want the planet to preview with a higher detail, you can click the **Update LOD** button, which will update it an additional **PreviewSteps** amount of times. You can also increase **PreviewSteps**. Every time you update a planet setting, it will be regenerated this amount of times.



SgtLandscapeColor

If your planet only has a heightmap and you want to color it based on height and slope, then you can use the **SgtLandscapeColor** component to color it. You can add this component to a child GameObject, or click the **Add Collider** button to do the same.

The **GradientTex** can be set to any texture with **read/write** enabled. This texture can be a picture of a landscape that has colors you want to use for your planet. You can find over 100 examples in the **Plugins/CW/Planet Forge/Gradients** folder.

SgtLandscapeDetail

To increase the surface detail of your planet, you can use the **SgtLandscapeDetail** component. You can add this component to a child GameObject, or click the **Add Detail** button to do the same.

SgtLandscapeBiome

If you want to combine multiple layers of detail and color, you should instead use the **SgtLandscapeBiome** component. You can add this component to a child GameObject, or click the **Add Biome** button to do the same. This has the same settings as **SgtLandscapeColor**, and the **Layers** setting allows up to 5 layers which have the same settings as the **SgtLandscapeDetail** component.

SgtLandscapeCollider

If you want your planet to have MeshColliders that match the visual mesh, then you can use the SgtLandscapeCollider component. You can add this component alongside the **SgtSphereLandscape** component, or click the **Add Collider** button to do the same. It will then generate colliders down to the specified **MinimumTriangleSize**.

SgtLandscapeObject

If you want to place objects on your planet's surface, you can add the **SgtLandscapeObject** to them. Your object's Transform component's **Position** and **Rotation** values will then be updated to match the underlying surface position and normal.

Space & MaskTex

The **SgtLandscapeColor/Detail/Biome** components all have the **Space** and **MaskTex** setting. If Space is set to **Global**, then the component will apply changes to the whole planet. If it's set to **Local**, then it will apply based on the current Transform position/rotation/scale, and you can see a preview of it in the **Scene** tab.

The **MaskTex** can be set to any **Single Channel** texture with the **Alpha8/R8** format, and it must have **Read/Write** enabled. When **Space** is set to **Global**, this mask will wrap around the whole planet using cylindrical (equirectangular) projection, the same as the base planet's AlbedoTex/HeightTex. If it's set to Local, then the mask will stretch to the size of the GameObject, which you can see a preview of in the **Scene** tab.

SgtSky

All planet atmospheres are made with the **SgtSky** component, which you can add to a new GameObject. This component has the **InnerMeshRadius** setting, which should match or be close to your **SgtSphereLandscape** component's **Radius** setting. The **Height** setting can be used to control the distance between the planet and the outer edge of the atmosphere.

This atmosphere effect requires camera depth data, which can be enabled by adding the **CwDepthTextureMode** component to your camera(s).

To control the overall sky color, you can add the **SgtSkyDepthTex** component alongside **SgtSky**.

If you set the **Lighting setting** to **Simple** or **Scatter**, then you can add the **SgtSkyLightingTex** component alongside **SgtSky**, which allows you to control the sunset color and position.

If you set the **Lighting setting** to **Scatter**, then you can add the **SgtSkyScatteringTex** component alongside **SgtSky**, which allows you to control the atmospheric scattering color of the sunlight as it goes through the atmosphere.

Planet Size

The planets in the example scenes have a radius of 500 units/meters. However, you can adjust the **SgtSphereLandscape** component's **Radius** setting to any value you like depending on your project requirements. If you change the radius by a large amount, it's likely the surface detail scale will be incorrect. To fix this, you can adjust the **SgtSphereLandscape** component's **SurfaceScale** component. For example, if you make the planet have a radius of 5000 (100x), then you may want to set **SurfaceScale** to 10 (10x) or greater.

If your planet is large (e.g. a radius of over 5000 units), then keep in mind the floating point precision of the camera rendering and physics will reduce and may cause issues. To fix this, your project must use some type of origin shifting system, where the objects get moved around the camera, rather than the camera moving around the objects. This results in the camera not moving too far from the scene origin (0,0,0), or not moving at all in some implementations. If you have the full Space Graphics Toolkit asset, then you can implement the Universe feature with the SgtFloatingCamera and SgtFloatingObject components, which adds origin shifting.

Known Issues

- If your planets are small, in some scenarios there are single-pixel gaps that can flicker between the terrain LOD chunks.
- The LOD checks are currently run in Update, so they have greater performance impact than is necessary, especially when near the planet surface when there are more LOD chunks.
- The SgtLandscapeDetail/Biome components with Space = Local don't take rotation into account when applying normal data to the planet, so the lighting direction may render incorrectly.
- The atmosphere component (SgtSky) appears to begin rendering incorrectly as the planet radius approaches or exceeds 5,000,000, or if the camera's Clipping Planes / Far approaches or exceeds 10,000,000.
- Clouds behind terrain can have a strong edge/outline. This is because by default the clouds are rendered at ¼ resolution, and the silhouette between the clouds and terrain are at a lower resolution. To improve the quality, you can change the SgtVolumeManager component's Downscale value to 1.
- Clouds can have an animated grainy look. This is because they use dithering to optimize performance by requiring fewer samples.. To improve the quality, you can change the SgtVolumeManager component's Downscale value to 1. You can also apply some type of post process anti-aliasing.
- If you enter the volumetric clouds of one planet, clouds of other distant planets will disappear. This is because they are both rendered to the same buffer and the closest one will override the other. Fixing this is possible, but it will incur a performance penalty.

These issues are currently being worked on and have the highest priority.

Clouds (Work in Progress)

Version 1.1.0 of Planet Forge introduces the SgtCloud component, which allows you to add a layer of volumetric clouds and cloud shadows around your planets.

To use it, you can add the **SgtCloud** component to an empty GameObject, and add this alongside your SgtSky GameObject.

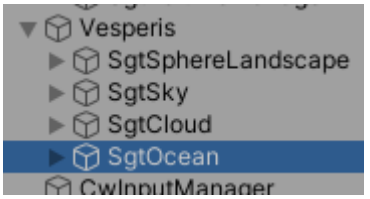
Next, drag and drop the **SgtCloud** GameObject into the **SgtSky** component's **Clouds** setting.

The camera in your scene must have the **SgtVolumeCamera** component.

Your scene must contain the **SgtVolumeManager** component.

NOTE: Clouds can only be rendered in one camera.
NOTE: Your SgtSky atmosphere must have a **Radius + Height** value greater than the SgtCloud's **Radius + Height**.

Ocean (Work in Progress)



Version 1.2.0 of Planet Forge introduces the SgtOcean component, which allows you to add a layer of volumetric ocean to your planets.

To use it, you can add the **SgtOcean** component to an empty GameObject, and add this alongside your SgtSky GameObject.

Next, drag and drop the **SgtOcean** GameObject into the **SgtSky** component's **Ocean** setting.

You can also add the **SgtOceanRays** component alongside the SgtOcean component. This will add light rays that shine through the ocean surface.

You can also add the **SgtOceanDebris** component alongside the SgtOcean component. This will add marine snow like effect that drifts under the surface of the ocean.

NOTE: If you want to detect if a point is under water, you can call the **SgtOcean** component's **CalcuclateWorldAltitude** method, which will return above 0 if the input point (e.g. transform.position) is above the ocean surface, or below 0 if it's inside the ocean.