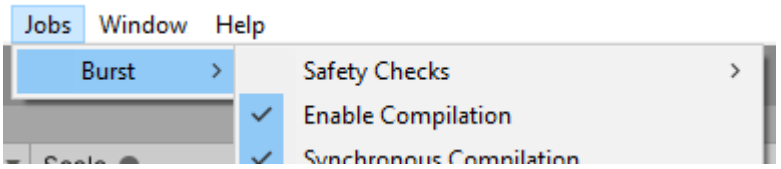
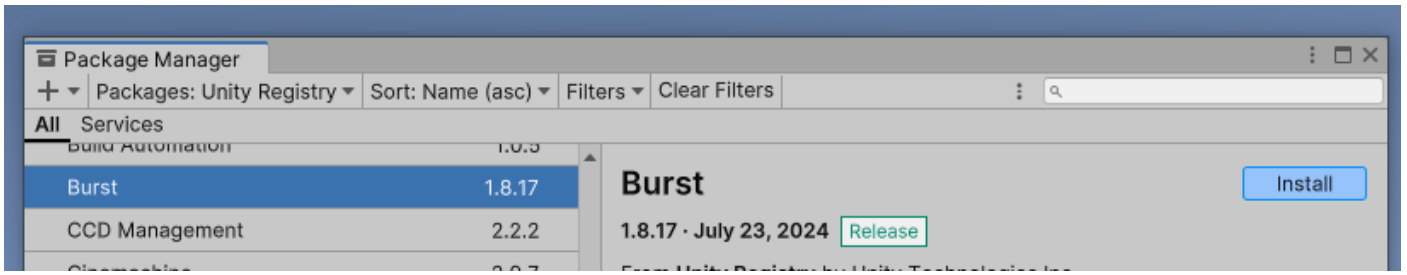
Planet Forge

Documentation

Dependencies

Before using Planet Forge, you must install the **Burst and Collections** packages from the **Window / Package Manager** inside Unity. Newer versions of Unity may automatically have these installed. Once installed, make sure Burst is 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 looks like. If you want to make your own planet, I recommend you duplicate one of these scenes and modify the values one at a time to see what happens.

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 from scratch, begin by right clicking in the **Hierarchy** tab, and select the **"Space Graphics Toolkit / Landscape / Sphere"** option. This will add a new GameObject to your scene with the **SgtSphereLandscape** component, which will add a sphere to your scene whose mesh detail automatically increases when the Main Camera approaches it. You can adjust the **Radius** setting as you wish.

Next, you can add a layer of detail to it by clicking the **"Add Detail"** button, which will add a child GameObject with the **SgtLandscapeDetail** component. The most important settings for this component are **HeightRange** and **GlobalSize**. **HeightRange** is the maximum height displacement for this layer. **GlobalSize** is the size of the detail texture in local space. For example, if your planet has a circumference of 1000 units and your GlobalSize is 100, then this layer of detail will tile 10 times around the planet. Keep in mind the final tiling values are snapped to avoid seams, so minor adjustments to the GlobalSize value may not visually change the planet.

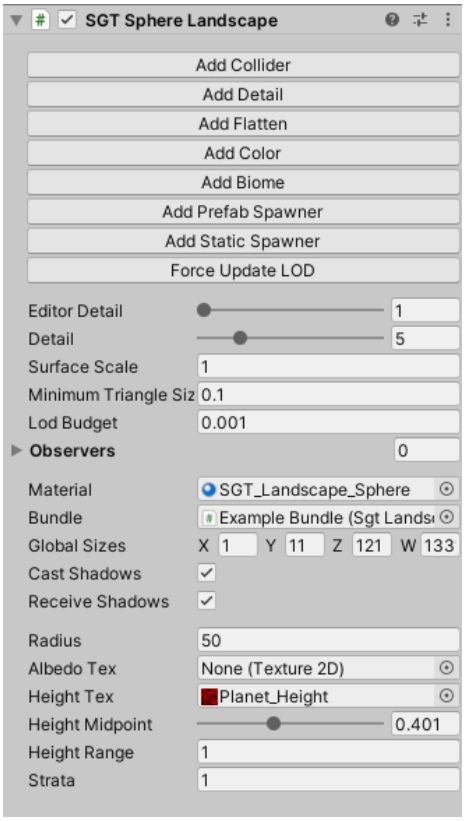
Next, to color the planet you can go back to the **SgtSphereLandscape** GameObject and click the **"Add Color"** button. This will add a new child GameObject with the **SgtLandscapeColor** component. The most important settings for this component are **Variation** and **Strata**, which you can experiment with. Keep in mind this color component overrides all previous colors, so you can't stack these unless you use a mask or apply it locally.

You may notice the SgtLandscapeDetail component has the **HeightIndex** setting, or how the **SgtLandscapeColor** component has **GradientIndex** setting. This controls which detail texture is used by this component, but by default there is only one detail or gradient texture. To change or add more textures, you must create your own texture bundle. To do this, go to your **SgtSphereLandscape** GameObject, and add the **SgtLandscapeBundle** component, and drag and drop this component into the **SgtSphereLandscape** component's **Bundle** setting. Once added, you can add any heightmap you like to the **HeightTextures** list, or any gradient texture you like to the **GradientTextures** list, etc.

To add an atmosphere to your planet, in the **Hierarchy** tab, you can right click the SgtSphereLandscape GameObject and select the **"Space Graphics Toolkit / Sky"** option. This will add a child GameObject with the **SgtSky** component. Feel free to adjust the **InnerMeshRadius** setting to match your planet, and adjust the **Height** setting based on how thick you want the atmosphere to be.

For more advanced features like ocean and clouds, I recommend you look at the example scenes in the **Plugins/CW/Planet Forge/Scenes** folder.

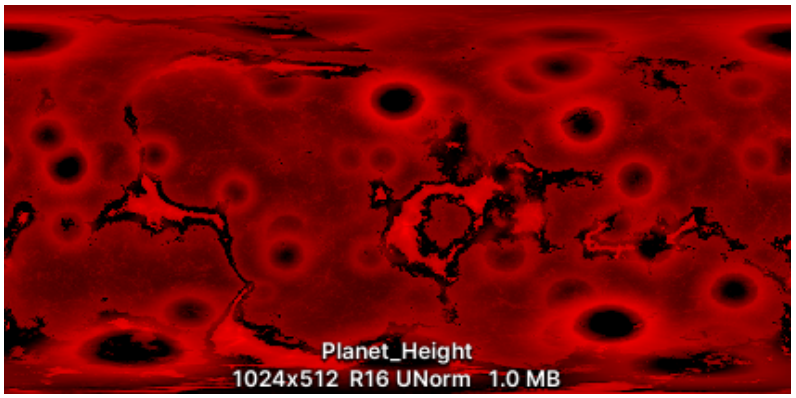
SgtSphereLandscape



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 **Observers** (default = MainCamera) approach. 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. While in edit mode, the planet will render with LOD fixed to the **EditorDetail** setting. If you increase this then the editor may become slow.

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 a **Single Channel Red** texture with **Read/Write** enabled.

All landscape modifier components use textures specified in the **Bundle** setting. If you want to use different textures, add the **SgtLandscapeBundle** component, and drop it into the **Bundles** setting. You can then add your own detail textures, gradient textures, etc.



SgtLandscapeColor

If you want to color your planet 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 Color** button to do the same.

The **GradientIndex** allows you to specify which gradient texture is used for the coloring. Gradient textures are set via the **SgtSphereLandscape** component's **Bundle** setting. 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.

You can also use the **Variation**, **Offset**, **Strata** and **Smooth** settings to fine tune the colors.

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.

The **HeightIndex** allows you to specify which height texture is used for the detail. Height textures are set via the **SgtSphereLandscape** component's **Bundle** setting. You can find over 100 examples in the **Plugins/CW/Planet Forge/Heightmaps** folder.

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** list has the same settings as the **SgtLandscapeDetail** component. Combining them into one makes it easier to adjust settings and copy between planets.

Space

The **SgtLandscapeColor/Detail/Biome** components all have the **Space** setting. When set to **Global**, it 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. You can see a preview of it in the **Scene** tab.

Mask

The **SgtLandscapeColor/Detail/Biome** components all have the **Mask** setting. When enabled, it will restrict where the changes are applied. For example, this can be used to apply colors to the planet's poles. Mask textures are set via the **SgtSphereLandscape** component's **Bundle** setting.

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**.

SgtLandscapePrefabSpawner

If you want to procedurally spawn prefabs on your planet, then you can use the SgtLandscapePrefabSpawner component. You can add this component alongside the **SgtSphereLandscape** component, or click the **Add Prefab Spawner** button to do the same. It will then spawn prefabs on terrain chunks whose triangles match the specified **Triangle Size**.

SgtLandscapeStaticSpawner

Spawning prefabs can be slow. If you want to spawn lots of basic objects like rocks and grass then you want to use the static spawner. You can add this component alongside the **SgtSphereLandscape** component, or click the **Add Static Spawner** button to do the same. This will spawn objects with the specified **Mesh** with the specified **Material**. Keep in mind you cannot use normal materials with this component, you must use materials with the **SGT / LandscapeStatic** shader.

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.

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 **SgtLandscapeDetail/Biome** component's **SurfaceScale** component. For example, if you make the planet have a radius of 5000 (10x), then you may want to set **SurfaceScale** to 10 (10x) or greater. Keep in mind that scaling the biomes up will not increase the final detail, so you may also want to add an additional layer of detail with a smaller GlobalSize.

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

- 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, especially when they are rolled around the local Z axis.

- 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, change the Smooth setting, or increase the SgtSky component's Detail setting. Keep in mind these changes will all reduce performance, so you must balance performance with quality.

- 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 in a future version, 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.

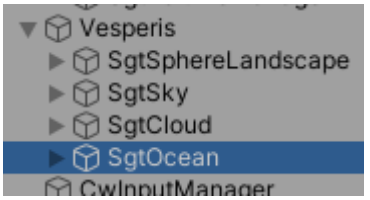
Next, drag and drop the **SgtCloud** GameObject into the **SgtSky** component's **Clouds** setting.

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: If you want cloud shadows to appear on the planet surface, drag and drop the **SgtCloud** GameObject into the **SgtSphereLandscape** component's **CloudShadow** setting.
NOTE: If you want cloud shadows to appear on the ocean surface, drag and drop the **SgtCloud** GameObject into the **SgtOcean** component's **CloudShadow** setting.
NOTE: If your clouds appear grainy, you can increase the **SgtSky** component's **Detail** setting, or set the **SgtVolumeManager** component's **Downscale** setting to 1, or enable the **Smooth** setting.

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.