GRIFFIN

SCRIPTING API

(v2.4.6)

INTRODUCTION

This document contains detail of the scripting API that Polaris V2 provide so you can use them to create your custom solutions.

The scripting reference is organised according to the classes available to scripts which are described along with their methods, properties and any other information relevant to their use.

API are grouped by namespaces they belong to, and can be selected from the Outline to the left.

Note that Editor and internal API are not documented and the whole API may be changed in future releases.

Pinwheel.Griffin

GStylizedTerrain

Represent the low poly terrain in the scene, responsible for receiving callbacks and hook up other process like geometry generation, foliage rendering, etc.

`public class GStylizedTerrain : MonoBehaviour`

Inheritance: MonoBehaviour → GStylizedTerrain

Static Properties

ActiveTerrains	A collection of active terrains in the scene.

Static Methods

ConnectAdjacentTiles	Connect all active terrains in the scene with their nearby, if AutoConnect is enabled.
Raycast	Cast a ray against all active terrain, filter by group ID.
MatchEdges	Remove geometry seam of all active terrain in a specific group ID.

Properties

TerrainData	The associate terrain data asset.
TopNeighbor	The neighbor terrain on the top side (+Z).
BottomNeighbor	The neighbor terrain on the bottom side (-Z).
LeftNeighbor	The neighbor terrain on the left side (-X).
RightNeighbor	The neighbor terrain on the right side (+X).
GroupId	An integer indicate the terrain group for auto connect and multi-tiles editing.

AutoConnect	Should it automatically connect with nearby terrains when needed?
Bounds	Bounding box in world space.

Methods

ConnectNeighbor	Connect the terrain to its nearby if AutoConnect is enable.
GetChunks	Get all terrain chunk objects of the current terrain.
Raycast	Cast a ray against the current terrain.
WorldPointToUV	Convert a world position into terrain UV (XZ01) space.
WorldPointToNormalized	Convert a world position into terrain normalized (XYZ01) space.
NormalizedToWorldPoint	Convert a point from terrain normalized (XYZ01) to world space.
GetWorldToNormalizedMatrix	Retrieve a matrix to transform point from world to normalized (XYZ01) space.
ForceLOD	Force the terrain to render geometry at a specific LOD.
GetHeightMapSample	Get a sample at a UV position from the height map.
GetInterpolatedHeightMapSample	Get a sample from the height map and blend with neighbor terrains at the edges.
UpdateTreesPosition	Snap all trees to the surface.
UpdateGrassPatches	Snap all grass instances to the surface and re-combine their meshes.
GetHeightMap	Get a render texture of the height map, for real-time processing.
GetSharpNormalMap	Get a render texture of the sharp-edged normal map, for real-time processing.
GetInterpolatedNormalMap	Get a render texture of the smooth-edged normal map, for real-time processing.
GetPerPixelNormalMap	Get a render texture of the per-pixel normal map, which derives from the height map, for real-time processing.

Refresh	Refresh geometry, in case of reference lost.

Delegates

PreProcessHeightMap	An event which is called before geometry generation, for creating custom effect on the height map.
PostProcessHeightMap	An event which is called after geometry generation, for reverse/restore the custom effect applied.

Details

`static IEnumerable<GStylizedTerrain> ActiveTerrains { get; }`

Retrieve a collection of all active terrains in the scene to iterate through, mostly useful for multi-terrain editing.

Example:

```
public static void ConnectAdjacentTiles()
{
    IEnumerator<GStylizedTerrain> terrains = ActiveTerrains.GetEnumerator();
    while (terrains.MoveNext())
    {
        GStylizedTerrain t = terrains.Current;
        t.ConnectNeighbor();
    }
}
```

`static void ConnectAdjacentTiles()`

Connect all active terrains in the scene to their nearby, if AutoConnect is enabled. Note that the terrain without terrain data will be ignore. Terrains which close enough to others will be connected.

`static bool Raycast(Ray ray, out RaycastHit hit, float distance, int groupId)`

Cast a ray against all active terrains in the scene, filter by groupId.

Parameters:

- ray: the ray to cast to the scene.
- hit: contains raycast result, if it hit something.
- distance: maximum distance for the ray.
- groupId: an integer to filter out which terrain will be in the test. Use -1 to include all terrains.

Return:

- bool: true if the ray hit any terrain.

Example:

```
Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
RaycastHit hit = new RaycastHit();
if (GStylizedTerrain.Raycast(ray, out hit, 1000, -1))
{
    DrawCursor(hit, true);
    Paint(hit);
}
else
{
    DrawCursor(hit, false);
}
```

`static void MatchEdges(int groupId)`

Remove geometry seam of active terrains in the group specified by group Id.

Parameters:

- groupId: an integer indicates the terrain group. Use -1 to include all groups.

GTerrainData TerrainData { get; set; }

The data asset associated with this terrain.

GStylizedTerrain TopNeighbor { get; set; }

GStylizedTerrain BottomNeighbor { get; set; }

GStylizedTerrain LeftNeighbor { get; set; }

GStylizedTerrain RightNeighbor { get; set; }

Access to the neighbor terrains on the 4 edges. Note that when connecting 2 terrains, you have to set the neighbor correctly on both of them. For example:

```
GStylizedTerrain t0;
GStylizedTerrain t1;
t0.LeftNeighbor = t1;
t1.RightNeighbor = t0;
```

---

int GroupId { get; set; }

An integer indicate the terrain group for auto connect and multi-tiles editing. Note that you have to assign a non-negative number to this property.

For some function, set groupId argument to -1 will iterate and process all terrains in the scene.

---

bool AutoConnect { get; set; }

Set this property to true will connect the current terrain to its nearby when calling ConnectAdjacentTiles() and ConnectNeighbor().

`Bounds Bounds { get; }`

Bounding box of the terrain in world space.

Example:

```
public static bool IsTerrainInsideFrustum(GStylizedTerrain t, Camera cam)
{
    Plane[] frustum = GeometryUtility.CalculateFrustumPlanes(cam);
    return GeometryUtility.TestPlanesAABB(frustum, t.Bounds);
}
```

`void ConnectNeighbor()`

Connect the terrain to its nearby. This function do nothing if Terrain Data is null or Auto Connect is set to false.

It can't connect 2 terrains if they're too far away. The distance threshold is equal to the terrain Width and Length.

`GTerrainChunk[] GetChunks()`

Retrieve a collection of children chunk game objects of the terrain.

Return:

- GTerrainChunk[]: collection of chunks.

Example:

```
public void ForceLOD(int level)
{
    GTerrainChunk[] chunks = GetChunks();
    for (int i = 0; i < chunks.Length; ++i)
    {
        chunks[i].LodGroupComponent.ForceLOD(level);
    }
}
```

`bool Raycast(Ray ray, out RaycastHit hit, float distance)`

Cast a ray against the terrain.

Parameters:

- ray: the ray to cast into the scene.
- hit: contains raycast result, if it hit the terrain.
- distance: maximum distance or the ray.

Return:

- bool: true if it hit the terrain.

`bool Raycast(Vector3 normalizePoint, out RaycastHit hit)`

Cast a top-down ray against the terrain, with the ray origin in normalized space 01.

Parameters:

- normalizePoint: the ray origin in normalized space. Note the only x and z component will be used to convert to world space origin.
- hit: contains raycast result, if it hit the terrain.

Return:

- bool: true if it hit the terrain.

Example:

```
public void SnapToTerrainCenter(GStylizedTerrain terrain)
{
    Vector3 normalizePos = new Vector3(0.5f, 0, 0.5f);
    RaycastHit hit;
    if (terrain.Raycast(normalizePos, out hit))
    {
        transform.position = hit.point;
    }
}
```

`Vector2 WorldPointToUV(Vector3 point)`

Convert a world position into terrain UV (XZ-plane, from 0 to 1) space. Useful when you want to sample terrain texture from a world position (from Raycast result).

Parameters:

- point: the point to convert in world space.

Return:

- Vector2: the UV coordinate.

Example:

```
public float GetHeightSampleAtWorldPos(GStylizedTerrain t, Vector3 p)
{
    if (t.TerrainData == null)
        return 0;
    Vector2 uv = t.WorldPointToUV(p);
    Color color = t.TerrainData.Geometry.HeightMap.GetPixelBilinear(uv.x, uv.y);
    return color.r;
}
```

`Vector3 WorldPointToNormalized(Vector3 point)`

Convert a point from world space to terrain normalized (0-1) space. Useful for adding foliage instances since they store position in normalized space.

Parameters:

- point: the point in world space to convert.

Return:

- Vector3: the point in normalized space.

Example:

```
public void AddTreeAtRaycastHit(GStylizedTerrain terrain, RaycastHit hit)
{
    if (terrain == null || terrain.TerrainData == null)
        return;

    int prototypeIndex = 0;
    GTreeInstance tree = GTreeInstance.Create(prototypeIndex);
    tree.Position= terrain.WorldPointToNormalized(hit.point);
    tree.Rotation = Quaternion.identity;
    tree.Scale = Vector3.one;
    terrain.TerrainData.Foliage.TreeInstances.Add(tree);
}
```

`Vector3 NormalizedToWorldPoint(Vector3 normalizedPoint)`

Convert a point in terrain normalized space to world space.

Parameters:

- normalizedPoint: the point to convert.

Return:

- Vector3: the converted point in world space.

`Matrix4x4 GetWorldToNormalizedMatrix()`

Retrieve a matrix to transform point from world to normalized space.

Return:

- Matrix4x4: the transform matrix.

`void ForceLOD(int level)`

Force the terrain to render geometry at a specific LOD.

Parameters:

- level: The level of detail to show.

`Vector4 GetHeightMapSample(Vector2 uv)`

Get pixel data of the height map at a specific UV location.

Parameters:

- uv: The UV coordinate to sample.

Return:

- Vector4: The bilinear interpolated data. This will return Vector4.zero if Terrain Data is null.

`Vector4 GetInterpolatedHeightMapSample(Vector2 uv)`

Get pixel data of the height map at a specific UV location. This data will be seamlessly smooth out between neighbor terrain at the edges.

Parameters:

- uv: the location to sample.

Return:

- Vector4: The bilinear interpolated data.

`void UpdateTreesPosition(bool showProgressBar = false)`

Snap all tree instances onto the terrain surface. Note that it only update trees lie within dirty regions set by SetTreeRegionDirty function (See example).

Parameters:

- showProgressBar: set to true if you want to show the progress bar in the editor (Editor only)

Example:

```
public void UpdateTerrainPartially(GStylizedTerrain t)
{
    Color[] oldHeightMapData = t.TerrainData.Geometry.HeightMap.GetPixels();
    //----------
    //Do some modification to the height map here
    //----------
    Color[] newHeightMapData = t.TerrainData.Geometry.HeightMap.GetPixels();

    //Determine which regions of the heightmap has been changed
    //This will split the terrain into a 5x5 rectangle grid and compare
    IEnumerable<Rect> dirtyRects = GCommon.CompareHeightMap(5, oldHeightMapData,
newHeightMapData);

    //Mark geometry as dirty for re-generate at those regions
    //Use an unit rect to update the whole terrain.
    t.TerrainData.Geometry.SetRegionDirty(dirtyRects);
    //Hook up the dirty callback for re-generating
    t.TerrainData.SetDirty(GTerrainData.DirtyFlags.Geometry);
    //Clean up
    t.TerrainData.Geometry.ClearDirtyRegions();

    //Mark trees and grasses as dirty at those regions
    //Use an unit rect to update the whole terrain.
    t.TerrainData.Foliage.SetTreeRegionDirty(dirtyRects);
    t.TerrainData.Foliage.SetGrassRegionDirty(dirtyRects);
    //Re-calculate position and batching
    t.UpdateTreesPosition();
    t.UpdateGrassPatches();
    //Clean up
    t.TerrainData.Foliage.ClearTreeDirtyRegions();
    t.TerrainData.Foliage.ClearGrassDirtyRegions();
}
```

`void UpdateGrassPatches(int prototypeIndex = -1, bool showProgressBar = false)`

Snap all grass instances onto the terrain surface and re-batch their meshes. Note that it only update the regions that are marked as dirty by SetGrassRegionDirty() function.

Parameters:

- prototypeIndex: the grass prototype to update. Use -1 to update all prototypes.
- showProgressBar: set to true to display the progress bar in the Editor.

Example:

```
public void UpdateTerrainPartially(GStylizedTerrain t)
{
    Color[] oldHeightMapData = t.TerrainData.Geometry.HeightMap.GetPixels();
    //----------
    //Do some modification to the height map here
    //----------
    Color[] newHeightMapData = t.TerrainData.Geometry.HeightMap.GetPixels();

    //Determine which regions of the heightmap has been changed
    //This will split the terrain into a 5x5 rectangle grid and compare
    IEnumerable<Rect> dirtyRects = GCommon.CompareHeightMap(5, oldHeightMapData,
newHeightMapData);

    //Mark geometry as dirty for re-generate at those regions
    //Use an unit rect to update the whole terrain.
    t.TerrainData.Geometry.SetRegionDirty(dirtyRects);
    //Hook up the dirty callback for re-generating
    t.TerrainData.SetDirty(GTerrainData.DirtyFlags.Geometry);
    //Clean up
    t.TerrainData.Geometry.ClearDirtyRegions();

    //Mark trees and grasses as dirty at those regions
    //Use an unit rect to update the whole terrain.
    t.TerrainData.Foliage.SetTreeRegionDirty(dirtyRects);
    t.TerrainData.Foliage.SetGrassRegionDirty(dirtyRects);
    //Re-calculate position and batching
    t.UpdateTreesPosition();
    t.UpdateGrassPatches();
    //Clean up
    t.TerrainData.Foliage.ClearTreeDirtyRegions();
    t.TerrainData.Foliage.ClearGrassDirtyRegions();
}
```

`RenderTexture GetHeightMap(int resolution)`

`RenderTexture GetSharpNormalMap(int resolution)`

`RenderTexture GetInterpolatedNormalMap(int resolution)`

`RenderTexture GetPerPixelNormalMap(int resolution)`

Get a copy of terrain height map or normal map in a Render Texture, for real-time processing such as texture generation and procedural texturing.

Parameters:

- resolution: size of the render texture.

Return:

- RenderTexture: the copied texture.

`void Refresh()`

Refresh geometry, in case of reference lost.

Example:

```
public static GStylizedTerrain CreateTerrain(GTerrainData data)
{
    GameObject g = new GameObject("Styllized Terrain");
    g.transform.localPosition = Vector3.zero;
    g.transform.localRotation = Quaternion.identity;
    g.transform.localScale = Vector3.one;
    GStylizedTerrain terrain = g.AddComponent<GStylizedTerrain>();
    terrain.GroupId = 0;
    terrain.TerrainData = data;
    terrain.Refresh();

    return terrain;
}
```

`delegate void HeightMapProcessCallback(Texture2D heightmap);`

`event HeightMapProcessCallback PreProcessHeightMap;`

`event HeightMapProcessCallback PostProcessHeightMap;`

Events which is called before and after geometry generation, useful for applying special effect on the height map.

For detail usage, please see the file GHeightMapFilter.cs in the package.

GTerrainChunk

Represent a chunk of a terrain.

Due to the limitation of vertex limit of a mesh (65000), a terrain will be splitted into many chunk, each chunk is responsible for generating geometry mesh at a specific region.

```
public class GTerrainChunk : MonoBehaviour
```

Inheritance: MonoBehaviour → GTerrainChunk

Static Methods

GetChunkMeshName	Return a name to load geometry mesh from generated resources.

Properties

Terrain	The parent terrain.
MeshFilterComponent	The MeshFilter component.
MeshRendererComponent	The MeshRenderer component.
MeshColliderComponent	The MeshCollider component.
LodGroupComponent	The LodGroup component.
Index	Index of the chunk in the chunk grid.

Methods

GetUvRange	Return the region the chunk covers in UV space.
Refresh	Refresh component and geometry references.

Detail

`static string GetChunkMeshName(Vector2 index, int lod)`

Return a name to load geometry mesh from generated resources.

Parameters:

- index: index of the chunk.
- lod: which LOD to get.

Return:

- string: the mesh name.

Example:

```
public Mesh GetMesh(GTerrainChunk chunk, int lod)
{
    string key = GetChunkMeshName(chunk.Index, lod);
    Mesh m = chunk.Terrain.TerrainData.GeometryData.GetMesh(key);
    return m;
}
```

`GStylizedTerrain Terrain { get; }`

The parent terrain this chunk belongs to.

`MeshFilter MeshFilterComponent { get; }`

`MeshRenderer MeshRendererComponent { get; }`

`MeshCollider MeshColliderComponent { get; }`

`LODGroup LodGroupComponent { get; }`

The components associated with this chunk for rendering and physics.

`Vector2 Index { get; }`

Index of the chunk in the chunk grid.

`Rect GetUvRange()`

Return a rectangle in UV space represent the region this chunk covers.

Return:

- Rect: the UV rectangle.

`void Refresh()`

Refresh component and geometry references.

GTerrainData

An asset type contains data associated with a terrain such as dimension, height map, albedo map, trees, etc.

This is the main point to take action on when creating your custom tools.

```
public class GTerrainData : ScriptableObject
```

Inheritance: ScriptableObject → GTerrainData

Static Delegates

GlobalDirty	Invoke when any terrain data in the project is marked as dirty.

Properties

Id	A unique identifier for the asset.
Geometry	Contains settings for terrain geometry such as dimension, height map, etc.
Shading	Contains settings for terrain shading/material such as splats prototype asset, material properties, etc.
Rendering	Contains settings for terrain rendering such as shadow, instancing, render distance, etc.
Foliage	Contains settings for foliage such as foliage prototype assets, foliage instances, etc.
GeometryData	Contains geometry meshes.
FoliageData	Contains foliage meshes.

Methods

SetDirty	Mark the terrain data as dirty for updating the terrain.
CopyTo	Copy data to other terrain data asset.

Delegates

Dirty	Invoke when the terrain data is marked as dirty.

Detail

```
delegate void GlobalDirtyHandler(GTerrainData data, GTerrainData.DirtyFlags flag);
```

```
static event GlobalDirtyHandler GlobalDirty;
```

Invoke when any terrain data in the project is marked as dirty.

```
string Id { get; }
```

A unique identifier for the asset. This Id is mostly used for data export/import and backup system, to determine which data file belongs to it.

```
GGeometry Geometry { get; }
```

Contains settings for terrain geometry such as dimension, height map, etc. If you're making a geometry tool, this will be a good starting point.

This property will never be NULL, so you don't need to check for it. It will refer to the sub-asset named "Geometry" under the terrain data in the Project window.

```
GShading Shading { get; }
```

Contains settings for terrain shading/material such as splats prototype asset, material properties, etc. If you're making a coloring/texturing tool, this will be a good starting point.

This property will never be NULL, so you don’t need to check for it. It will refer to the sub-asset named “Shading” under the terrain data in the Project window.

`GRendering Rendering { get; }`

Contains settings for terrain rendering such as shadow, instancing, render distance, etc.

This property will never be NULL, so you don’t need to check for it. It will refer to the sub-asset named “Rendering” under the terrain data in the Project window.

`GFoliage Foliage { get; }`

Contains settings for foliage such as foliage prototype assets, foliage instances, etc. If you’re making a spawning tool, this will be a good starting point.

This property will never be NULL, so you don’t need to check for it. It will refer to the sub-asset named “Foliage” under the terrain data in the Project window.

`GTerrainGeneratedData GeometryData {get; }`

`GTerrainGeneratedData FoliageData { get; }`

Contains generated geometry and foliage meshes. Access these properties to query terrain mesh for rendering.

These properties will never be NULL, so you don’t need to check for it. They will refer to the assets named “GeneratedGeometry” and “GeneratedFoliage” in the Project window, usually next to the terrain data.

`void SetDirty(GTerrainData.DirtyFlags flag)`

Mark the terrain data as dirty for updating the terrain.

Parameters:

- flag: Indicate which part of the terrain should be update. Where:
    - DirtyFlags.Geometry: Re-generate geometry on main thread.
    - DirtyFlags.GeometryAsync: Re-generate geometry asynchronously.
    - DirtyFlags.Rendering: Update chunk renderer setting such as shadow, etc.
    - DirtyFlags.Shading: Update terrain material.
    - DirtyFlags.Foliage: Refresh tree prototype config, remove invalid foliage instances.
    - DirtyFlags.All: Update everything.

Note that for Geometry, GeometryAsync and Foliage flags, the terrain only update only the regions that is marked as dirty.

Example:

```
public void ResetGeometry(GStylizedTerrain terrain)
{
    Texture2D heightMap = terrain.TerrainData.Geometry.HeightMap;
    GCommon.FillTexture(heightMap, Color.clear);
    terrain.TerrainData.Geometry.SetRegionDirty(GCommon.UnitRect);
    terrain.TerrainData.SetDirty(GTerrainData.DirtyFlags.Geometry);
}
```

`void CopyTo(GTerrainData des)`

Copy settings to another terrain data asset.

This function does NOT copy terrain textures.

`delegate void DirtyHandler(GTerrainData.DirtyFlags flag);`

`event DirtyHandler Dirty;`

Invoke when the terrain data is marked as dirty for updating the terrain.

Example:

```
public GTerrainData terrainData;

private void OnEnable()
{
    if (terrainData != null)
    {
        terrainData.Dirty += OnTerrainDataDirty;
    }
}

private void OnDisable()
{
    if (terrainData != null)
    {
        terrainData.Dirty -= OnTerrainDataDirty;
    }
}

private void OnTerrainDataDirty(GTerrainData.DirtyFlags dirtyFlag)
{
    //DirtyFlags enum is a bitmask type, there can be multiple enum values
    //in a single variable, so it's a bit complicated to check for these
    //flags

    if ((dirtyFlag & GTerrainData.DirtyFlags.Geometry) ==
GTerrainData.DirtyFlags.Geometry)
    {
        //terrain data is marked as Geometry dirty
    }
    else if ((dirtyFlag & GTerrainData.DirtyFlags.GeometryAsync) ==
GTerrainData.DirtyFlags.GeometryAsync)
    {
        //terrain data is marked as GeometryAsync dirty
    }

    if ((dirtyFlag & GTerrainData.DirtyFlags.Shading) ==
GTerrainData.DirtyFlags.Shading)
    {
        //it may ALSO marked as Shading dirty
    }

    if ((dirtyFlag & GTerrainData.DirtyFlags.Rendering) ==
GTerrainData.DirtyFlags.Rendering)
    {
        //it may ALSO marked as Rendering dirty
    }

    if ((dirtyFlag & GTerrainData.DirtyFlags.Foliage) ==
GTerrainData.DirtyFlags.Foliage)
    {
        //it may ALSO marked as Foliage dirty
    }
}
```

GTerrainData.DirtyFlags

An enum type indicates which part of the terrain is dirty/modified and should be updated.

`public enum GTerrainData.DirtyFlags : byte`

Enum Values

None = 0	Nothing is dirty. This value is mostly unused.
Geometry = 1	Geometry is dirty, and should be re-generate on the main thread.
GeometryAsync = 2	Geometry is dirty, and should be re-generate on background threads.
Rendering = 4	Rendering settings such as shadow, instancing, render distance, etc. are dirty, chunk renderers should be updated.
Shading = 8	Material and textures are dirty, chunk renderers should be updated.
Foliage = 16	Trees and grasses are dirty, foliage renderer and prototype assets should be updated.
All = 255	Everything should be updated.

Detail

This enum is provided as a bit mask field, so many flags may present in a single variable. To check for a flag, some bitwise operations need to be performed.

For example:

```
public GTerrainData terrainData;

private void OnEnable()
{
    if (terrainData != null)
    {
        terrainData.Dirty += OnTerrainDataDirty;
    }
}

private void OnDisable()
{
    if (terrainData != null)
    {
        terrainData.Dirty -= OnTerrainDataDirty;
    }
}

private void OnTerrainDataDirty(GTerrainData.DirtyFlags dirtyFlag)
{
    //DirtyFlags enum is a bitmask type, there can be multiple enum values
    //in a single variable, so it's a bit complicated to check for these
    //flags

    if ((dirtyFlag & GTerrainData.DirtyFlags.Geometry) ==
GTerrainData.DirtyFlags.Geometry)
    {
        //terrain data is marked as Geometry dirty
    }
    else if ((dirtyFlag & GTerrainData.DirtyFlags.GeometryAsync) ==
GTerrainData.DirtyFlags.GeometryAsync)
    {
        //terrain data is marked as GeometryAsync dirty
    }

    if ((dirtyFlag & GTerrainData.DirtyFlags.Shading) ==
GTerrainData.DirtyFlags.Shading)
    {
        //it may ALSO marked as Shading dirty
    }

    if ((dirtyFlag & GTerrainData.DirtyFlags.Rendering) ==
GTerrainData.DirtyFlags.Rendering)
    {
        //it may ALSO marked as Rendering dirty
    }

    if ((dirtyFlag & GTerrainData.DirtyFlags.Foliage) ==
GTerrainData.DirtyFlags.Foliage)
    {
        //it may ALSO marked as Foliage dirty
    }
}
```

GGeometry

An asset type contains data related to terrain geometry. This is the sub-asset of the terrain data in the Project window.

```
public class GGeometry : ScriptableObject
```

Inheritance: ScriptableObject → GGeometry

Constants

HEIGHT_MAP_NAME	Name of the Height Map texture.

Static Properties

HeightMapFormat	Texture format of the Height Map (RGBA32)
HeightMapRTFormat	Texture format of the Height Map render texture (ARGB32)
PolygonProcessorTypeCollection	A list of types that implement IPolygonProcessor

Properties

TerrainData	The parent terrain data it belongs to.
Width	Size of the terrain on X-axis.
Height	Size of the terrain on Y-axis.
Length	Size of the terrain on Z-axis
HeightMapResolution	Size of the height map in pixel.
HeightMap	The height map texture.
MeshBaseResolution	Minimum resolution (sub-division level) of terrain meshes.
MeshResolution	Maximum resolution (sub-division level) of terrain meshes.
ChunkGridSize	Size of the terrain chunk grid. Total number of chunk is

	ChunkGridSize squared.
LODCount	Number of LOD to generate. Maximum is 4.
DisplacementSeed	A random seed to displace vertex position on XZ-plane.
DisplacementStrength	The amount to displace vertex position on XZ-plane.
PolygonProcessorName	Name of the polygon processor being in used.

Methods

ResetFull	Reset the asset to it default value, including textures.
CleanUp	Remove redundant data such as terrain meshes that will never be used.
SetRegionDirty	Mark a particular region of the terrain as dirty for re-generating.
GetDirtyRegions	Retrieve all dirty regions.
ClearDirtyRegions	Clear/Unmark all dirty regions.
CopyTo	Copy data to another instance.
GetDecodedHeightMapSample	Sample the height map at a specific UV location and decode the data.

Detail

GGeometry is one of the main classes to interact with when working on terrain geometry such as change terrain size, re-generate terrain mesh, etc.

Note that the terrain will stay unchanged when setting these properties, until you officially call SetRegionDirty(rect) on this object and SetDirty(flags) on the terrain data. For example, the code for changing terrain size should look like this:

```
public void SetTerrainSize(GStylizedTerrain terrain, Vector3 size)
{
    terrain.TerrainData.Geometry.Width = size.x;
    terrain.TerrainData.Geometry.Height = size.y;
    terrain.TerrainData.Geometry.Length = size.z;

    //mark the whole terrain as dirty
    terrain.TerrainData.Geometry.SetRegionDirty(new Rect(0, 0, 1, 1));
    //invoke the Dirty callback
    //the GStylizedTerrain component will receive and re-generate geometry
    terrain.TerrainData.SetDirty(GTerrainData.DirtyFlags.Geometry);
    //terrain geometry is fresh now, we can clean up dirty regions
    //to avoid unnecessary computation next time
    terrain.TerrainData.Geometry.ClearDirtyRegions();
}
```

`const string HEIGHT_MAP_NAME = "Height Map";`

Name of the Height Map texture.

`static TextureFormat HeightMapFormat { get; }`

Format of the Height Map texture. This will return RGBA32.

`static RenderTextureFormat HeightMapRTFormat { get; }`

Format of the Height Map render texture. This is mainly use for real-time painting.

`static List<Type> PolygonProcessorTypeCollection { get; }`

A list of types that implement IPolygonProcessor interface.

`GTerrainData TerrainData { get; }`

The parent terrain data it belongs to.

Use this property to access to other terrain data modules such as GShading, GRendering and GFoliage.

`float Width { get; }`

`float Height { get; }`

`float Length { get; }`

Size of the terrain in 3 dimensions. The minimum value for them is Vector3(1, 0, 1). Note that change these value won't change the terrain until you invoke the Dirty callback.

Example:

```
public void SetTerrainSize(GStylizedTerrain terrain, Vector3 size)
{
    terrain.TerrainData.Geometry.Width = size.x;
    terrain.TerrainData.Geometry.Height = size.y;
    terrain.TerrainData.Geometry.Length = size.z;

    //mark the whole terrain as dirty
    terrain.TerrainData.Geometry.SetRegionDirty(new Rect(0, 0, 1, 1));
    //invoke the Dirty callback
    //the GStylizedTerrain component will receive and re-generate geometry
    terrain.TerrainData.SetDirty(GTerrainData.DirtyFlags.Geometry);
    //terrain geometry is fresh now, we can clean up dirty regions
    //to avoid unnecessary computation next time
    terrain.TerrainData.Geometry.ClearDirtyRegions();
}
```

`int HeightMapResolution { get; set; }`

Size of the Height Map in pixel.

You are freely to set any value for this property, however, it will clamp the value to the closest-power-of-two number from 1 to 8192. For most case, a resolution of 512 or 1024 is enough for a terrain.

When you set this property to another value, data from the height map will be resample. This is a destructive process. For example: set resolution from 512 down to 256, then go back to 512, will give you a blurry image.

`Texture2D HeightMap { get; }`

Return reference to the height map texture. This texture can also be accessed from the Project window, under the terrain data asset.

Data in the height map are stored as 4 channel, where:

- RG: Elevation/height value (0-1), encoded.
- B: Additional/manual sub-division value. This value is used to add more detail to a particular region of the terrain manually. The interval for each sub-division level is 0.1. For example: a value of 0.5 will add 5 more sub-divisions to the surface.
- A: Holes/Visibility value. A value greater than 0.5 will strip of the polygon at that location.

You can use GetPixels() and SetPixels() to manipulate the texture. Remember to call Apply() and hook up the Dirty callback to re-generate the terrain.

Manipulate the height map on the GPU for real-time editing is out of scope for this document, please see the GElevationPainter.cs file to learn more.

See EncodeFloatRG, DecodeFloatRG for more information.

Example:

```
public void FlattenHeightmap(GStylizedTerrain terrain, int startX, int startY, int
width, int length, float elevation)
{
    Texture2D hm = terrain.TerrainData.Geometry.HeightMap;
    Color[] data = hm.GetPixels();

    int endX = startX + width - 1;
    int endY = startY + length - 1;
    int resolution = terrain.TerrainData.Geometry.HeightMapResolution;
    Vector2 enc = Vector2.zero;
    for (int x = startX; x <= endX; ++x)
    {
        for (int y = startY; y <= endY; ++y)
        {
            //beware: the pixel location may be outside of texture dimension,
adding some check!

            int pixelIndex = y * resolution + x;
            Color c = data[pixelIndex];
            enc = GCommon.EncodeFloatRG(elevation);
            c.x = enc.x;
            c.y = enc.y;
            data[pixelIndex] = c;
        }
    }

    hm.SetPixels(data);
    hm.Apply();

    //calculate the dirty region, or just use a unit rect to update the whole
terrain (slower)
    Vector2 rectPos = new Vector2(
        startX / resolution,
        startY / resolution);
    Vector2 rectSize = new Vector2(
        width / resolution,
        length / resolution);
    Rect rect = new Rect(rectPos, rectSize);

    //update geometry
    terrain.TerrainData.Geometry.SetRegionDirty(rect);
    terrain.TerrainData.SetDirty(GTerrainData.DirtyFlags.Geometry);
    terrain.TerrainData.Geometry.ClearDirtyRegions();

    //update tree and grass
    terrain.TerrainData.Foliage.SetTreeRegionDirty(rect);
    terrain.TerrainData.Foliage.SetGrassRegionDirty(rect);
    terrain.UpdateTreesPosition();
    terrain.UpdateGrassPatches();
    terrain.TerrainData.Foliage.ClearTreeDirtyRegions();
    terrain.TerrainData.Foliage.ClearGrassDirtyRegions();
}
```

`int MeshBaseResolution { get; set; }`

`int MeshResolution { get; set; }`

Minimum (0-10) and maximum (0-15) resolution (sub-division level) of terrain meshes.

When setting these properties, you have to invoke the Dirty callback to re-generate terrain mesh.

`int ChunkGridSize { get; set; }`

Size if the terrain chunk grid. For example, setting this property to 5 will create a 5x5 grid, equivalent to 25 chunks game object.

The ideal value for this is not to exceed 8, because higher value consume way more memory on generation and issue more draw calls when rendering the terrain.

You have to invoke the Dirty callback to apply change to the terrain.

`int LODCount { get; set; }`

Number of LOD to generate. Maximum is 4 levels.

This value should be leave at 1 when editing the terrain for faster performance and less memory consumption.

You have to invoke the Dirty callback to apply change to the terrain.

`int DisplacementSeed { get; set; }`

`float DisplacementStrength { get; set; }`

Random seed and intensity (0-1) for vertex displacement on XZ-plane.

Use these properties to break the uniform look of terrain meshes.

You have to invoke the Dirty callback to apply change to the terrain.

`string PolygonProcessorName { get; set; }`

Name of the polygon processor class being in used.

Polygon processor is a class that implement IGPolygonProcessor interface, to inject custom processing for each triangle of the terrain on generation such as adding custom tesselation, convert albedo to vertex color, etc.

Built-in values are:

- "GMidpointSplitPolygonProcessor"
- "GAlbedoToVertexColorPolygonProcessor"
- "GAlbedoToSharpVertexColorPolygonProcessor"

See IGPolygonProcessor interface and GPolygon class for more information.

`void ResetFull()`

Reset the asset to it default value, including height map color.

You don't need to invoke the Dirty callback, this function apply change automatically to terrain geometry.

`void CleanUp()`

Remove redundant data such as terrain meshes that will never be used.

For example, geometry meshes of LOD3 will never be used if you reduce LODCount to a value less than 4.

This function take action to the terrain meshes in your asset. You don't need to invoke the Dirty callback.

`void SetRegionDirty(Rect uvRect)`

Mark a particular region of the terrain as dirty for re-generating.

Parameters:

- uvRect: a rectangle in UV space represent the dirty region.

Example:

```
public void FlattenHeightmap(GStylizedTerrain terrain, int startX, int startY, int
width, int length, float elevation)
{
    Texture2D hm = terrain.TerrainData.Geometry.HeightMap;
    Color[] data = hm.GetPixels();

    int endX = startX + width - 1;
    int endY = startY + length - 1;
    int resolution = terrain.TerrainData.Geometry.HeightMapResolution;

    for (int x = startX; x <= endX; ++x)
    {
        for (int y = startY; y <= endY; ++y)
        {
            //beware: the pixel location may be outside of texture dimension,
adding some check!

            int pixelIndex = y * resolution + x;
            Color c = data[pixelIndex];
            c.r = elevation;
            data[pixelIndex] = c;
        }
    }

    hm.SetPixels(data);
    hm.Apply();

    //calculate the dirty region, or just use a unit rect to update the whole
terrain (slower)
```

```
    Vector2 rectPos = new Vector2(
        startX / resolution,
        startY / resolution);
    Vector2 rectSize = new Vector2(
        width / resolution,
        length / resolution);
    Rect rect = new Rect(rectPos, rectSize);

    //update geometry
    terrain.TerrainData.Geometry.SetRegionDirty(rect);
    terrain.TerrainData.SetDirty(GTerrainData.DirtyFlags.Geometry);
    terrain.TerrainData.Geometry.ClearDirtyRegions();

    //update tree and grass
    terrain.TerrainData.Foliage.SetTreeRegionDirty(rect);
    terrain.TerrainData.Foliage.SetGrassRegionDirty(rect);
    terrain.UpdateTreesPosition();
    terrain.UpdateGrassPatches();
    terrain.TerrainData.Foliage.ClearTreeDirtyRegions();
    terrain.TerrainData.Foliage.ClearGrassDirtyRegions();
}
```

`void SetRegionDirty(IEnumerable<Rect> uvRects)`

Mark some regions of the terrain as dirty for re-generating.

Parameters:

- uvRects: the collection of dirty rectangle in UV space.

`Rect[] GetDirtyRegions()`

Retrieve all dirty regions.

Return:

- Rect[]: an array contains all dirty regions.

`void ClearDirtyRegions()`

Clear/Unmark all dirty regions.

`void CopyTo(GGeometry des)`

Copy data to another instance.

`Vector4 GetDecodedHeightMapSample(Vector2 uv)`

Sample the height map at a specific UV location and decode the data.

The decoded data is as follow:

- X: normalized height value (0-1)
- Y: the same value as X
- Z: additional sub-division
- A: visibility

GShading

An asset type contains data related to terrain shading and material. This is the sub-asset of the terrain data in the Project window.

`public class GShading : ScriptableObject`

Inheritance: ScriptableObject → GShading

Constants

ALBEDO_MAP_NAME	Name of the Albedo map.
METALLIC_MAP_NAME	Name of the Metallic map.
COLOR_BY_HEIGHT_MAP_NAME	Name of the Color By Height map.
COLOR_BY_NORMAL_MAP_NAME	Name of the Color By Normal map.
COLOR_BLEND_MAP_NAME	Name of the Color Blend map.
SPLAT_CONTROL_MAP_NAME	Name of the Splat Control maps. This name will be appended with control map index.

Properties

TerrainData	The terrain data asset it belongs to.
MaterialToRender	The current material being in used for rendering.
CustomMaterial	User-assigned material.
AlbedoMapResolution	Size of the Albedo map in pixel.
AlbedoMap	The Albedo map texture.
MetallicMapResolution	Size of the Metallic map in pixel.
MetallicMap	The Metallic map texture.
AlbedoMapPropertyName	Name of the Albedo map in shader property to bind to.
MetallicMapPropertyName	Name of the Metallic map in shader property to bind to.

ColorByHeight	A gradient to defined terrain color based on vertex height.
ColorByNormal	A gradient to defined terrain color based on vertex normal.
ColorBlendCurve	A curve to interpolate between ColorByHeight and ColorByNormal based on vertex height.
ColorByHeightPropertyName	Name of the Color By Height texture in shader property to bind to.
ColorByNormalPropertyName	Name of the Color By Normal texture in shader property to bind to.
ColorBlendPropertyName	Name of the Color Blend texture in shader property to bind to.
DimensionPropertyName	Name of the Dimension vector in shader property to bind to.
ColorByHeightMap	The Color By Height texture. Generated from Color By Height gradient.
ColorByNormalMap	The Color By Normal texture. Generated from Color By Normal gradient.
ColorBlendMap	The Color Blend texture. Generated from Color Blend curve.
Splats	The splats prototype group asset.
SplatControlResolution	Size of the Splat Control maps in pixel.
SplatsControls	The Splat Control maps.
SplatControlMapPropertyName	Name of the Splat Control textures in shader property to bind to. This name will be appended with splat control index.
SplatMapPropertyName	Name of the Splat texture in shader property to bind to. This name will be appended with splat index.
SplatNormalPropertyName	Name of the Splat normal texture in shader property to bind to. This name will be appended with splat index.
SplatMetallicPropertyName	Name of the Splat metallic value in shader property to bind to. This name will be appended with splat index.

SplatSmoothnessPropertyName	Name of the Splat smoothness value in shader property to bind to. This name will be appended with splat index.
SplatControlMapCount	Total number of Splat Control maps.

Methods

ResetFull	Reset the asset to its default value, including textures.
UpdateMaterials	Update material settings and references.
UpdateLookupTextures	Update ColorByHeight, ColorByNormal and ColorBlend textures.
GetSplatControl	Return a splat control texture at index.
ConvertSplatsToAlbedo	Render all splat textures to Albedo map with correct blending.
CopyTo	Copy settings to other instance.

Detail

GShading is one of the main classes to interact with when working on terrain shading and material such as paint color and textures, etc.

Note that for the terrain material to be set up and render correctly, you have to call UpdateMaterials() on this object and SetDirty(flags) on the terrain data. For example, the code for changing terrain material should look like this:

```
public void SetTerrainMaterial(GStylizedTerrain terrain, Material mat)
{
    terrain.TerrainData.Shading.CustomMaterial = mat;
    //update material configuration
    terrain.TerrainData.Shading.UpdateMaterials();
    //update material reference for all terrain chunks
    terrain.TerrainData.SetDirty(GTerrainData.DirtyFlags.Shading);
}
```

`const string ALBEDO_MAP_NAME = "Albedo Map";`

`const string METALLIC_MAP_NAME = "Metallic Map";`

`const string COLOR_BY_HEIGHT_MAP_NAME = "Color By Height Map";`

`const string COLOR_BY_NORMAL_MAP_NAME = "Color By Normal Map";`

`const string COLOR_BLEND_MAP_NAME = "Color Blend Map";`

`const string SPLAT_CONTROL_MAP_NAME = "Splat Control Map";`

Name of the terrain textures in Project window.

`GTerrainData TerrainData { get; }`

The terrain data asset it belongs to.

`Material MaterialToRender { get; }`

The material being in used for rendering. This will return the CustomMaterial at the moment.

`Material CustomMaterial { get; set; }`

The material assigned by user.

Note that to configure apply this material for the terrain, you have to call UpdateMaterials() function on this object and invoke the Dirty callback on the terrain asset.

Example:

```
public void SetTerrainMaterial(GStylizedTerrain terrain, Material mat)
{
    terrain.TerrainData.Shading.CustomMaterial = mat;
    //update material configuration
    terrain.TerrainData.Shading.UpdateMaterials();
    //update material reference for all terrain chunks
    terrain.TerrainData.SetDirty(GTerrainData.DirtyFlags.Shading);
}
```

`int AlbedoMapResolution { get; set; }`

Size of the Albedo map in pixel. This property will be clamped to the closest-power-of-two value in range of 1-8192.

When setting this value, the Albedo map will be resampled.

`Texture2D AlbedoMap { get; }`

Return a reference to the Albedo texture.

You can use GetPixels() and GetPixels() to manipulate this texture. Remember to call Apply(), UpdateMaterials() and invoke the Dirty callback to refresh terrain rendering.

Manipulate this texture on the GPU is out of scope for this document, please see the GAlbedoPainter.cs file to learn more.

`int MetallicMapResolution { get; set; }`

Size of the Metallic map in pixel. This property will be clamped to the closest-power-of-two value in range of 1-8192.

When setting this value, the Metallic map will be resampled.

`Texture2D MetallicMap { get; }`

Return a reference to the Metallic texture.

You can use GetPixels() and GetPixels() to manipulate this texture. Remember to call Apply(), UpdateMaterials() and invoke the Dirty callback to refresh terrain rendering.

Manipulate this texture on the GPU is out of scope for this document, please see the GMetallicPainter.cs file to learn more.

`string AlbedoMapPropertyName { get; set; }`

`string MetallicMapPropertyName { get; set; }`

`string ColorByHeightPropertyName { get; set; }`

`string ColorByNormalPropertyName { get; set; }`

`string ColorBlendPropertyName { get; set; }`

`string DimensionPropertyName { get; set; }`

`string SplatControlMapPropertyName { get; set; }`

`string SplatMapPropertyName { get; set; }`

`string SplatNormalPropertyName { get; set; }`

`string SplatMetallicPropertyName { get; set; }`

`string SplatSmoothnessPropertyName { get; set; }`

Name of shader properties to bind textures/vector to. You only care about them if you're making custom shader for your terrain, otherwise just leave them as default.

For example, if your shader name the Albedo map as "_MainTex", then you must set AlbedoMapPropertyName to "_MainTex".

For splat shader, the property name will be appended with texture/value index. For example, if you name your shader property as "_Control0", "_Control1", "_Splat0", "_Splat1", "_Normal0", "_Normal1", "_Metallic0", "_Metallic1", "_Smoothness0", "_Smoothness1" and so on, then your property name config should look like this:

- SplatControlMapPropertyName: "_Control"
- SplatMapPropertyName: "_Splat"
- SplatNormalPropertyName: "_Normal"
- SplatMetallicPropertyName: "_Metallic"
- SplatSmoothnessPropertyName: "_Smoothness"

Remember to call UpdateMaterials() and invoke the Dirty callback to refresh terrain rendering.

`Gradient ColorByHeight { get; set; }`

`Gradient ColorByNormal { get; set; }`

`AnimationCurve ColorBlendCurve { get; set; }`

Gradients and curve to define terrain color based on vertex height and normal vector. These properties only take effect when using Gradient Lookup shader variant.

After modifying these property, you have to call UpdateLookupTextures() to turn them into textures.

Remember to call UpdateMaterials() and invoke the Dirty callback to refresh terrain rendering.

`Texture2D ColorByHeightMap { get; }`

`Texture2D ColorByNormalMap { get; }`

`Texture2D ColorBlendMap { get; }`

Textures that are generated from ColorByHeight, ColorByNormal gradient and ColorBlend curve.

`GSplatPrototypeGroup Splats { get; set; }`

The splats prototype group asset assigned to this terrain. This asset contains data about splat layers like texture, normal map, tiling, offset, metallic, smoothness, etc.

This asset may be shared across multiple terrains, be careful when modifying it.

See GSplatPrototype and GSplatPrototypeGroup for more detail.

Remember to call UpdateMaterials() and invoke the Dirty callback to refresh terrain rendering.

Example:

```
public void ImportSplatTextures(GStylizedTerrain terrain, Texture2D[] layers)
{
    List<GSplatPrototype> prototypes = new List<GSplatPrototype>();
    for (int i = 0; i < layers.Length; ++i)
    {
        GSplatPrototype p = new GSplatPrototype();
        p.Texture = layers[i];
        p.NormalMap = null;
        p.TileOffset = new Vector2(0, 0);
        p.TileSize = new Vector2(10, 10);
        p.Metallic = 0;
        p.Smoothness = 0;
        prototypes.Add(p);
    }

    terrain.TerrainData.Shading.Splats.Prototypes = prototypes;

#if UNITY_EDITOR
    UnityEditor.EditorUtility.SetDirty(terrain.TerrainData.Shading.Splats);
#endif
}
```

`int SplatControlResolution { get; set; }`

Size of the Splat Control maps in pixel. This value will be clamped to the closest-power-of-two in range of 1-8192.

When setting this value, the Splat Control maps will be resampled.

Remember to call UpdateMaterials() and invoke the Dirty callback to refresh terrain rendering.

`Texture2D[] SplatControls { get; }`

Return all Splat Control textures. The number of control texture is determined by the number of splat prototype in your splat asset. Each 4 layers will create 1 control texture.

See SplatControlMapCount for more detail.

To access to a splat control map at index, it's better to use GetSplatControl(index) function.

`int SplatControlMapCount { get; }`

Return the number of control maps needed for blending all splat layers. Each 4 layers will use 1 control map.

`void ResetFull()`

Reset the asset to its default value, including textures.

This function refresh terrain rendering automatically.

`void UpdateMaterials()`

Update material settings and references.

This function will iterate through all of your material properties name, then bind that property to the appropriate texture.

For splat related property, the property name will be appended with its texture/layer index.

`void UpdateLookupTextures()`

Update ColorByHeight, ColorByNormal and ColorBlend textures.

Use this function after modifying lookup gradients and curve.

Remember to call UpdateMaterials() and invoke the Dirty callback to refresh terrain rendering.

`Texture2D GetSplatControl(int index)`

Get access to a splat control texture at specified index.

Parameters:

- index: index of the control texture to retrieve.

Return:

- Texture2D: reference to the control texture.

After modifying the texture, remember to call UpdateMaterials() and invoke the Dirty callback to refresh terrain rendering.

Example:

```
public void ModifySplatControls(GStylizedTerrain terrain)
{
    GShading shading = terrain.TerrainData.Shading;
    int controlMapCount = shading.SplatControlMapCount;
    for (int i = 0; i < controlMapCount; ++i)
    {
        Texture2D controlMap = shading.GetSplatControl(i);
        //---------
        //Do some work on the controlMap here
        //---------
        controlMap.Apply();
    }

    shading.UpdateMaterials();
    terrain.TerrainData.SetDirty(GTerrainData.DirtyFlags.Shading);
}
```

void ConvertSplatsToAlbedo()

Render all splat textures to Albedo map with correct blending.

---

void CopyTo(GShading des)

Copy settings to other instance.

Parameters:

- des: the instance to copy to.

GRendering

An asset type contains data related to terrain rendering such as shadow, instancing, render distance, etc. This is the sub-asset of the terrain data in the Project window.

`public class GRendering : ScriptableObject`

Inheritance: ScriptableObject → GRendering

Properties

TerrainData	The terrain data asset it belongs to.
CastShadow	Should the terrain geometry cast shadow?
ReceiveShadow	Should the terrain geometry receive shadow?
DrawFoliage	Should it render trees and grasses or not?
EnableInstancing	Toggle GPU Instancing for trees.
BillboardStart	The distance from the camera where it begin to render trees as billboards.
TreeDistance	Maximum distance to render trees.
GrassDistance	Maximum distance to render grasses.

Methods

ResetFull	Reset the asset to its default values.
CopyTo	Copy data to another instance.

Detail

GRendering is one of the main classes to control terrain rendering (geometry and foliage). For shadow option, you have to invoke the Dirty callback to refresh terrain chunk rendering.

Example:

```
public void DisableShadow(GStylizedTerrain terrain)
{
    terrain.TerrainData.Rendering.CastShadow = false;
    terrain.TerrainData.Rendering.ReceiveShadow = false;
    terrain.TerrainData.SetDirty(GTerrainData.DirtyFlags.Rendering);
}
```

`GTerrainData TerrainData { get; }`

The terrain data asset it belongs to.

`bool CastShadow { get; set; }`

`bool ReceiveShadow { get; set; }`

Toggle shadow mode for terrain geometry.

You have to invoke the Dirty callback for the terrain chunks to reflect the change.

Example:

```
public void DisableShadow(GStylizedTerrain terrain)
{
    terrain.TerrainData.Rendering.CastShadow = false;
    terrain.TerrainData.Rendering.ReceiveShadow = false;
    terrain.TerrainData.SetDirty(GTerrainData.DirtyFlags.Rendering);
}
```

`bool DrawFoliage { get; set; }`

Turn on or off the foliage renderer. Useful when you want to use different foliage system.

`bool EnableInstancing { get; set; }`

Toggle GPU instancing for trees.

The tree material must have Instancing enable, if not, it will fall back to non-instancing rendering.

Instancing is not available in the Scene View.

`float BillboardStart { get; set; }`

`float TreeDistance { get; set; }`

`float GrassDistance { get; set; }`

Distance setting for trees and grasses rendering.

Value range:

- BillboardStart: 0-5000
- TreeDistance: 0-5000
- GrassDistance: 0-500

If your tree prototype doesn’t have a billboard asset assigned, consider to set the maximum value for BillboardStart so it barely rendered as billboard.

`void ResetFull()`

Reset the asset to its default values.

`void CopyTo(GRendering des)`

Copy data to another instance.

Parameters:

- des: the instance to copy data to.

GFoliage

An asset type contains data related to terrain foliage such as tree prototypes, grass prototypes, foliage instances, etc. This is the sub-asset of the terrain data in the Project window.

`public class GFoliage : ScriptableObject`

Inheritance: ScriptableObject → GFoliage

Properties

TerrainData	The terrain data asset it belongs to.
Trees	The tree prototype group asset assigned to this terrain.
TreeInstances	All tree instances on the terrain.
TreeSnapMode	Determine how tree should snap to the surface.
TreeSnapLayerMask	A mask to filter which collider the trees can snap on.
Grasses	The grass prototype group asset assigned to this terrain.
GrassSnapMode	Determine how grass should snap to the surface.
GrassSnapLayerMask	A mask to filter which collider the grasses can snap on.
PatchGridSize	Size of the grass patch grid. Total grass patch is equivalent to PatchGridSize squared.
GrassPatches	Return the grass patch grid, flatten into a 1D array.
EstimatedGrassStorageMB	Size of grass meshes stored on hard disk in megabyte.
EnableInteractiveGrass	Toggle interactive/touch bending grass feature.
VectorFieldMapResolution	Size of the interactive grass vector field in pixel.
BendSensitive	How fast the grass bend when touching the player.
RestoreSensitve	How fast the grass restore to it original shape when the player leaved.
GrassInstanceCount	Total number of grass instances.

Methods

ResetFull	Reset the asset to its default values.
Refresh	Refresh foliage prototype configuration and remove invalid instances.
AddGrassInstances	Add new grass instances to the terrain and put them to their correct patch.
GetGrassInstances	Return a copy of all grass instances.
ClearGrassInstances	Remove all grass instances from the terrain.
CleanUp	Remove all unused grass patch meshes.
SetTreeRegionDirty	Mark a region as dirty for updating trees.
GetTreeDirtyRegions	Retrieve all regions that is marked as tree dirty.
ClearTreeDirtyRegions	Clear all tree dirty regions.
SetGrassRegionDirty	Mark a region as dirty for updating grasses.
GetGrassDirtyRegions	Retrieve all regions that is marked as grass dirty.
ClearGrassDirtyRegions	Clear all grass dirty regions.
CopyTo	Copy data to other instance.

Detail

GFoliage is one of the main classes to interact with when working on spawning foliage onto terrain surface.

Different from Unity built-in terrain, tree and grass (detail) prototype are assigned using the asset type name GTreePrototypeGroup and GGrassPrototypeGroup, this way you can share the prototype setting between multiple terrain and it's suitable for multi-terrain editing.

Sometime you need to invoke the Dirty callback to refresh foliage config, but this step is not so critical.

However, it is important to mark terrain regions as dirty using SetTreeRegionDirty() and SetGrassRegionDirty() so that they can update correctly.

Example:

```
public void AddFoliage(GStylizedTerrain terrain, List<GTreeInstance> trees,
List<GGrassInstance> grasses)
{
    //Add new instances
    terrain.TerrainData.Foliage.TreeInstances.AddRange(trees);
    terrain.TerrainData.Foliage.AddGrassInstances(grasses);

    //Mark the whole terrain as dirty for both tree and grass
    terrain.TerrainData.Foliage.SetTreeRegionDirty(new Rect(0, 0, 1, 1));
    terrain.TerrainData.Foliage.SetGrassRegionDirty(new Rect(0, 0, 1, 1));

    //Snap trees to the surface
    terrain.UpdateTreesPosition();
    //Snap grasses to the surface and re-batch meshes
    terrain.UpdateGrassPatches();

    //Clean up
    terrain.TerrainData.Foliage.ClearTreeDirtyRegions();
    terrain.TerrainData.Foliage.ClearGrassDirtyRegions();
}
```

`GTerrainData TerrainData { get; }`

The terrain data asset it belongs to.

`GTreePrototypeGroup Trees { get; set; }`

The tree prototype group asset assigned to this terrain.

This asset may be shared between multiple terrains, so be careful when modifying it.

`List<GTreeInstance> TreeInstances { get; set; }`

All tree instances of the terrain.

Example:

```
public void AddFoliage(GStylizedTerrain terrain, List<GTreeInstance> trees,
List<GGrassInstance> grasses)
{
    //Add new instances
    terrain.TerrainData.Foliage.TreeInstances.AddRange(trees);
    terrain.TerrainData.Foliage.AddGrassInstances(grasses);

    //Mark the whole terrain as dirty for both tree and grass
    terrain.TerrainData.Foliage.SetTreeRegionDirty(new Rect(0, 0, 1, 1));
    terrain.TerrainData.Foliage.SetGrassRegionDirty(new Rect(0, 0, 1, 1));

    //Snap trees to the surface
    terrain.UpdateTreesPosition();
    //Snap grasses to the surface and re-batch meshes
    terrain.UpdateGrassPatches();

    //Clean up
    terrain.TerrainData.Foliage.ClearTreeDirtyRegions();
    terrain.TerrainData.Foliage.ClearGrassDirtyRegions();
}
```

`GSnapMode TreeSnapMode { get; set; }`

`GSnapMode GrassSnapMode { get; set; }`

Determine how tree & grass should snap to the surface: on the terrain itself or on world colliders.

Note that you have to call terrain.UpdateTreesPosition() or terrain.UpdateGrassPatches() for this to take effect.

`LayerMask TreeSnapLayerMask { get; set; }`

`LayerMask GrassSnapLayerMask { get; set; }`

Masks to filter out which world collider trees and grasses can snap on.

Note that you have to call terrain.UpdateTreesPosition() or terrain.UpdateGrassPatches() for this to take effect.

`GGrassPrototypeGroup Grasses { get; set; }`

The grass prototype group asset assigned to this terrain.

This asset may be shared between multiple terrain, so be careful when modifying it.

`int PatchGridSize { get; set; }`

Size of the grass patch grid. Total grass patch is equivalent to PatchGridSize squared.

When modifying this value, the patch collection will be re-sampled which will keep grass instances in their correct patch.

Higher value will increase grass density, but issue more draw calls.

You have to call terrain.UpdateGrassPatches() for this to re-batch grass meshes.

`GGrassPatch[] GrassPatches { get; }`

Return the grass patch grid, flatten into a 1D array.

`float EstimatedGrassStorageMB { get; }`

Size of grass meshes stored on hard disk in megabyte.

`bool EnableInteractiveGrass { get; set; }`

Toggle interactive/touch bending grass feature.

`int VectorFieldMapResolution { get; set; }`

Size of the interactive grass vector field in pixel.

`float BendSensitive { get; set; }`

How fast the grass bend when touching the player.

Value range: 0-1

`float RestoreSensitive { get; set; }`

How fast the grass restore to it original shape when the player leaved.

Value range: 0-1

`int GrassInstanceCount { get; }`

Total number of grass instances.

`void ResetFull()`

Reset the asset to its default values.

`void Refresh()`

Refresh foliage prototype configuration and remove invalid instances.

`void AddGrassInstances(List<GGrassInstance> instances)`

Grass instances are stored per-patch. This function will find the correct patch for each instance based on its position and add it to the list.

Parameters:

- instances: the instances to add to the terrain.

Example:

```
public void AddFoliage(GStylizedTerrain terrain, List<GTreeInstance> trees,
List<GGrassInstance> grasses)
{
    //Add new instances
    terrain.TerrainData.Foliage.TreeInstances.AddRange(trees);
    terrain.TerrainData.Foliage.AddGrassInstances(grasses);

    //Mark the whole terrain as dirty for both tree and grass
    terrain.TerrainData.Foliage.SetTreeRegionDirty(new Rect(0, 0, 1, 1));
    terrain.TerrainData.Foliage.SetGrassRegionDirty(new Rect(0, 0, 1, 1));

    //Snap trees to the surface
    terrain.UpdateTreesPosition();
    //Snap grasses to the surface and re-batch meshes
    terrain.UpdateGrassPatches();

    //Clean up
    terrain.TerrainData.Foliage.ClearTreeDirtyRegions();
    terrain.TerrainData.Foliage.ClearGrassDirtyRegions();
}
```

`List<GGrassInstance> GetGrassInstances()`

Return a copy list of all grass instances on the terrain.

To modify existing instance, you have to access to the instance list of an appropriate patch, copy the instance to temporary variable, modify it, then assign it back. For example:

```
public void ModifyExistingGrassInstance(GStylizedTerrain terrain)
{
    //To simplify the problem, we will ignore all exception risk

    //Refer to the first grass patch
    GGrassPatch patch = terrain.TerrainData.Foliage.GrassPatches[0];

    //Copy the first instance out
    GGrassInstance grass = patch.Instances[0];

    //Double it scale
    grass.Scale = 2 * grass.Scale;

    //Assign it back
    patch.Instances[0] = grass;

    //Re-batch grass mesh
    terrain.UpdateGrassPatches();
}
```

`void ClearGrassInstances()`

Remove all grass instances from the terrain.

`void CleanUp()`

Remove all unused grass patch meshes.

Grass patch meshes become redundant when you lower the PatchGridSize and in some other cases.

`void SetTreeRegionDirty(Rect uvRect)`

`void SetTreeRegionDirty(IEnumerable<Rect> uvRects)`

Mark a particular region of the terrain as dirty for updating trees.

Parameters:

- uvRect: the rectangle in UV space.
- uvRects: a collection of rectangles in UV space.

`Rect[] GetTreeDirtyRegions()`

Retrieve all regions that is marked as tree dirty.

Return:

- Rect[]: a collection of tree dirty regions.

`void ClearTreeDirtyRegions()`

Clear all tree dirty regions.

`void SetGrassRegionDirty(Rect uvRect)`

`void SetGrassRegionDirty(IEnumerable<Rect> uvRects)`

Mark a particular region on the terrain as dirty for updating grasses.

Parameters:

- uvRect: a rectangle in UV space.
- uvRects: a collection of rectangles in UV space.

`Rect[] GetGrassDirtyRegions()`

Retrieve all grass dirty regions.

Return:

- Rect[]: a collection of grass dirty regions.

`void ClearGrassDirtyRegions()`

Clear all grass dirty regions.

`void CopyTo(GFoliage des)`

Copy data to other instance.

Parameters:

- des: the instance to copy to.

GGrassPatch

Represent a patch, or a group of grass instances.

`public class GGrassPatch`

Static Methods

GetPatchMeshName	Return a name for fetching grass mesh from generated data.

Properties

Foliage	The terrain foliage setting asset it belongs to.
Index	Index of this patch in the patch grid.
Instances	Contains all grass instances on this patch.

Methods

GetUvRange	Return the UV rectangle that the patch cover. Useful for frustum culling or dirty test.
UpdateMeshes	Update patch meshes for all prototypes.
UpdateMesh	Update patch mesh for a specific prototype.
GetMesh	Return the grass mesh of a specific prototype.

Detail

`static string GetPatchMeshName(Vector2 index, int prototypeIndex)`

Return a name to fetch the patch mesh from generated data.

Parameters:

- index: index of the patch in the patch grid.
- prototypeIndex: index of the grass prototype.

Return:

- string: the key to fetch grass mesh.

`List<GGrassInstance> Instances { get; set; }`

Return a list of all grass instances in this patch.

To modify an existing grass instance, you have to copy the instance from the list, modify it, then assign it back. For example:

```
public void ModifyExistingGrassInstance(GStylizedTerrain terrain)
{
    //To simplify the problem, we will ignore all exception risk

    //Refer to the first grass patch
    GGrassPatch patch = terrain.TerrainData.Foliage.GrassPatches[0];

    //Copy the first instance out
    GGrassInstance grass = patch.Instances[0];

    //Double it scale
    grass.Scale = 2 * grass.Scale;

    //Assign it back
    patch.Instances[0] = grass;

    //Re-batch grass mesh
    terrain.UpdateGrassPatches();
}
```

To add new grass instances to the terrain, you're recommended to use GFoliage.AddGrassInstances instead.

GTerrainGeneratedData

An asset type contains terrain generated data such as geometry mesh and grass mesh. This is the asset next to the Terrain Data in the Project window when you first create the terrain.

```
public class GTerrainGeneratedData : ScriptableObject,
ISerializationCallbackReceiver
```

Inheritance: ScriptableObject → GTerrainGeneratedData

Properties

TerrainData	The terrain data it belongs to.

Methods

SetMesh	Add a mesh to the internal collection with a specified key, and save it to asset.
GetMesh	Return the mesh from the internal collection with a specific key.
DeleteMesh	Delete a mesh from the internal collection with a specific key.
GetKeys	Return all keys present in the internal collection.

Detail

This class is the container for all terrain generated stuff such as geometry meshes and grass meshes.

Each mesh will be associate with a key. To get the correct key to fetch the mesh, see GetChunkMeshName and GetPatchMeshName function.

GSplatPrototype

Contains information about a terrain splat layer.

`public class GSplatPrototype`

Properties

Texture	The diffuse texture.
NormalMap	The normal map.
TileSize	Size of a tile in world space.
TileOffset	Offset of a tile in world space.
Metallic	Metallic value, for PBR shading.
Smoothness	Smoothness value, for PBR shading.

Methods

Equals	Compare if the 2 prototype are identical.
CopyTo	Copy data to another instance.

Detail

This class describes a terrain splat layer. Different from Unity terrain, you cannot assign it directly to the terrain, but indirectly using a Splat Prototype Group.

Splat Prototype Group is an asset type contains a set of splat prototype, which can be shared between multiple terrain. This way it will suitable for multi-terrain editing.

Example: This code shows how to add a new splat prototype using a texture:

```
public void AddSplatLayer(GStylizedTerrain terrain, Texture2D tex)
{
    GSplatPrototype proto = new GSplatPrototype();
    proto.Texture = tex;
    proto.NormalMap = null;
    proto.TileSize = new Vector2(10, 10);
    proto.TileOffset = new Vector2(0, 0);
    proto.Metallic = 0;
    proto.Smoothness = 0;

    GSplatPrototypeGroup group = terrain.TerrainData.Shading.Splats;
    group.Prototypes.Add(proto);
    terrain.TerrainData.SetDirty(GTerrainData.DirtyFlags.Shading);
}
```

GSplatPrototypeGroup

An asset type contains a set of Splat Prototype. This asset can be shared between multiple terrains which suitable for multi-terrain editing.

```
public class GSplatPrototypeGroup : ScriptableObject
```

Inheritance: ScriptableObject → GSplatPrototypeGroup

Static Methods

Create	Create new instance using an array of Unity splat prototypes

Properties

Prototypes	A set of Griffin Splat Prototype.

Detail

```
static GSplatPrototypeGroup Create(SplatPrototype[] layers) //2018.1+
```

```
static GSplatPrototypeGroup Create(TerrainLayer[] layers) //2018.3+
```

Create new instance using a set of Unity splat prototypes. Useful when converting from Unity terrain asset.

Parameters:

- layers: the Unity splat prototypes set.

Return:

- GSplatPrototypeGroup: The new prototype group instance.

`List<GSplatPrototype> Prototypes { get; set; }`

Return the prototypes set.

GTreePrototype

Represent a kind of tree on the terrain.

`public class GTreePrototype`

Static Methods

Create	Create a new prototype using a game object.

Properties

Prefab	The source game object of the tree.
SharedMesh	The mesh to render the tree.
SharedMaterials	All materials of the tree.
ShadowCastingMode	Should it cast shadow?
ReceiveShadow	Should it receive shadow?
Layer	The layer to render the tree to.
KeepPrefabLayer	Whether to use custom layer or prefab layer to render the tree.
Billboard	The billboard asset assigned to this prototype.
HasCollider	Is there any capsule collider attached?
ColliderInfo	Information about the tree collider.
PivotOffset	A small offset on Y-axis when render the tree. Useful for quickly fix its position on the surface.
BaseRotation	Additional rotation to fix tree model alignment.
BaseScale	Additional scale to fix tree model unit scale.
IsValid	Is the prototype valid for rendering?

Methods

Refresh	Read the source prefab and refresh settings.

Detail

This class contains information about a kind of tree on the terrain. Different from Unity terrain, you cannot assign it directly to the terrain, but indirectly using a Tree Prototype Group, which can be shared between multiple terrain, this way it’s suitable for multi-terrain editing.

Example: This code shows how to add a new tree kind to the terrain using a prefab:

```
public void AddTreeType(GStylizedTerrain terrain, GameObject tree)
{
    GTreePrototype proto = GTreePrototype.Create(tree);
    GTreePrototypeGroup group = terrain.TerrainData.Foliage.Trees;
    group.Prototypes.Add(proto);
}
```

`static GTreePrototype Create(GameObject g)`

Create a new prototype using a prefab.

Parameters:

- g: the source prefab.

Return:

- GTreePrototype: the new prototype.

GTreePrototypeGroup

An asset type contains a set of Tree Prototype. This asset can be shared between multiple terrains which suitable for multi-terrain editing.

`public class GTreePrototypeGroup : ScriptableObject`

Inheritance: ScriptableObject → GTreePrototypeGroup

Static Methods

Create	Create new instance using an array of Unity tree prototypes

Properties

Prototypes	A set of Griffin Tree Prototype.

Detail

`static GTreePrototypeGroup Create(TreePrototype[] treePrototypes)`

Create new instance using a set of Unity tree prototypes. Useful when converting from Unity terrain asset.

Parameters:

- treePrototypes: the Unity tree prototypes set.

Return:

- GTreePrototypeGroup: the new prototype group instance.

`List<GTreePrototype> Prototypes { get; set; }`

Return the tree prototypes set.

GTreeInstance

Represent a tree instance on the terrain.

`public struct GTreeInstance`

Properties

PrototypeIndex	Index of the prototype in the prototype group.
Position	Position of the tree in normalized (XYZ01) space.
Rotation	Rotation of the tree.
Scale	Scale of the tree.

Methods

Create	Create a new instance.

Detail

`static GTreeInstance Create(int prototypeIndex)`

Create a new instance.

Parameters:

- prototypeIndex: index of the prototype.

Return:

- GTreeInstance: the new instance.

Example:

```
public void AddTreeInstance(GStylizedTerrain terrain, int prototypeIndex)
{
    GTreeInstance tree = GTreeInstance.Create(prototypeIndex);
    terrain.TerrainData.Foliage.TreeInstances.Add(tree);
}
```

GGrassPrototype

Represent a kind of grass on the terrain.

`public class GGrassPrototype`

Static Methods

Create	Create a new prototype using a texture or a game object.

Properties

Texture	The texture to render.
Prefab	The source prefab, if the prototype is used as detail object.
Size	Size of the prototype.
Layer	The render layer of the prototype.
Shape	Shape of the grass: Quad, Cross, TriCross, etc.
CustomMesh	Custom shape of the grass.
DetailMesh	The mesh to render the prototype, if it’s used as detail object.
DetailMaterial	The material to render the prototype, if it’s used as detail object.
ShadowCastingMode	Should it cast shadow?
ReceiveShadow	Should it receive shadow?
AlignToSurface	Should it align to surface normal vector on batching. Useful for rock and pebbles, etc.
PivotOffset	A small offset on Y-axis to fix instance position.
BendFactor	How much it’s affected by wind?

Methods

GetBaseMesh	Return the mesh to render the instance.
Refresh	Refresh prefab references.

Detail

This class contains information about a kind of grass or detail object on the terrain. Different from Unity terrain, you cannot assign it directly to the terrain, but indirectly using a Grass Prototype Group, which can be shared between multiple terrain, this way it’s suitable for multi-terrain editing.

Example: This code shows how to add a new grass kind to the terrain using a texture:

```
public void AddGrassType(GStylizedTerrain terrain, Texture2D grass)
{
    GGrassPrototype proto = GGrassPrototype.Create(grass);
    GGrassPrototypeGroup group = terrain.TerrainData.Foliage.Grasses;
    group.Prototypes.Add(proto);
}
```

`static GGrassPrototype Create(Texture2D tex)`

`static GGrassPrototype Create(GameObject prefab)`

Create new instance using a texture or prefab. The corresponding shape are Quad and DetailObject.

Parameters:

- tex: The texture to render the grass.
- prefab: The source prefab for the detail object.

Return:

- GGrassPrototype: the new prototype.

GGrassPrototypeGroup

An asset type contains a set of Grass Prototype. This asset can be shared between multiple terrains which suitable for multi-terrain editing.

`public class GGrassPrototypeGroup : ScriptableObject`

Inheritance: ScriptableObject → GGrassPrototypeGroup

Static Methods

Create	Create new instance using an array of Unity detail prototypes

Properties

Prototypes	A set of Griffin Grass Prototype.

Detail

`static GGrassPrototypeGroup Create(DetailPrototype[] detailPrototypes)`

Create new instance using a set of Unity detail prototypes.

Parameters:

- detailPrototypes: the Unity detail prototypes set.

Return:

- GGrassPrototypeGroup: The new prototype group instance.

`List<GGrassPrototype> Prototypes { get; set; }`

Return the grass prototypes set.

GGrassInstance

Represent a grass instance on the terrain.

`public struct GGrassInstance`

Properties

PrototypeIndex	Index of the prototype in the prototype group.
Position	Position of the grass in normalized (XYZ01) space.
Rotation	Rotation of the grass.
Scale	Scale of the grass.

Methods

Create	Create a new instance.

Detail

`static GGrassInstance Create(int prototypeIndex)`

Create new instance.

Parameters:

- prototypeIndex: index of the prototype.

Return:

- GGrassInstance: the new instance.

Example:

```
public void AddGrassInstances(GStylizedTerrain terrain, int prototypeIndex)
{
    List<GGrassInstance> instances = new List<GGrassInstance>();
    for (int i = 0; i < 100; ++i)
    {
        GGrassInstance grass = GGrassInstance.Create(prototypeIndex);
        grass.Position = new Vector3(Random.value, 0, Random.value);
        instances.Add(grass);
    }
    terrain.TerrainData.Foliage.AddGrassInstances(instances);
}
```

GCommon

Contains some definition for common tasks.

`public static class GCommon`

Methods

EncodeFloatRG	Encode a normalized float value to a Vector2, mostly useful for height map manipulation.
DecodeFloatRG	Decode a Vector2 to a normalized float value, mostly useful for height map sampling.

Detail

`static Vector2 EncodeFloatRG(float v)`

Encode a normalized **[0-1)** float value to a Vector2, for storing height value in the height map. Note that 1.0 will not be encoded correctly, you need to clamp the value to something like 0.999999f for example.

See Height Map for more information.

`static float DecodeFloatRG(Vector2 enc)`

Decode a encoded Vector2 to normalized float value, for reading height value from the RG channel of the height map.

See Height Map for more information.

Pinwheel.Griffin.BackupTool

We’re updating this section, if you need more information about it, please contact us at support@pinwheel.studio.

Pinwheel.Griffin.BillboardTool

We’re updating this section, if you need more information about it, please contact us at support@pinwheel.studio.

Pinwheel.Griffin.DataTool

We’re updating this section, if you need more information about it, please contact us at support@pinwheel.studio.

Pinwheel.Griffin.ExtensionSystem

We’re updating this section, if you need more information about it, please contact us at support@pinwheel.studio.

Pinwheel.Griffin.GroupTool

We're updating this section, if you need more information about it, please contact us at support@pinwheel.studio.

Pinwheel.Griffin.HelpTool

We’re updating this section, if you need more information about it, please contact us at support@pinwheel.studio.

Pinwheel.Griffin.PaintTool

We’re updating this section, if you need more information about it, please contact us at support@pinwheel.studio.

Pinwheel.Griffin.SplineTool

We’re updating this section, if you need more information about it, please contact us at support@pinwheel.studio.

Pinwheel.Griffin.StampTool

We’re updating this section, if you need more information about it, please contact us at support@pinwheel.studio.

Pinwheel.Griffin.TextureTool

We’re updating this section, if you need more information about it, please contact us at support@pinwheel.studio.