Using Views for Rendering

The C# Client Library allows to integrate the Voxel Farm platform into existing rendering environments.

The VoxelFarmView object in the library has the concept of a “focus” scope. Typically, a 3D application will have a camera, and the camera is expected to move often unpredictably and expose different regions of the 3D content. The “focus” point for a Voxel Farm View is defined as a region of space the application wants to see in higher detail. The VoxelFarmView object will load higher resolution data in the focus region, and it will lower the resolution of the data as it becomes more distant from the focus scope.

The VoxelFarmView also natively exploits the spatial and temporal coherence of the spatial models. The data used for a particular focus setting could be mostly reusable from the next focus point. The library makes sure only the portions that changed or were not in focus before will be requested from the server and loaded. The library also keeps an internal ephemeral cache that will remember some regions of space for a while, since very often users return to a previous vintage point.

To best exploit these advantages, the application must provide a special interface to the view. This interface will be called by the view to notify new content is ready for use. This is the “IVoxelFarmRenderer” interface. The application must supply its own implementation of the interface by calling the “SetRenderer” method:

```
MyView.SetRenderer(myRenderer);
```

Implementing IVoxelFarmRenderer

The “IVoxelFarmRenderer” is used by the application to receive notifications from the C# Client. This section describes the methods of this interface. For an example of a working IVoxelFarmRenderer implementation, look at the VoxelFarmUnityView class in the Unity example.

```
void SetVoxelFarmOrigin(double x, double y, double z);
```

The client notifies the application the origin of the project in Voxel Farm coordinates.

```
IVoxelFarmCellResource BakeCellMesh(ushort clusterId, ulong cellId, byte layer, byte submesh, uint[] faces, float[] vertices, float[] normals, byte[] colors);
```

The client requests the application to create a mesh resource for a given section of the scene. The clusterId, cellId, layer and submesh arguments identify the mesh section. The faces, vertices,

normal and colors arrays contain the actual mesh information.

```
IVoxelFarmCellResource BakeCellTexture(ushort clusterId, ulong cellId, byte layer, byte textureType, ushort textureSize, byte[] textureData, byte textureFormat);
```

The client requests the application to create a texture resource for a given section of the scene. The clusterId, cellId, layer and textureType arguments identify the texture section. The textureSize argument specifies the dimensions of the texture in pixels. The textureData array contains the actual RGB information. The textureFormat argument specifies in which format the texture is stored.

```
IVoxelFarmCellResource BakeCellPoints(ushort clusterId, ulong cellId, byte layer, float[] vertices, byte[] colors, double[,] aabb);
```

The client requests the application to create a points resource for a given section of the scene. The clusterId, cellId and layer identify the point cloud section. The vertices and colors array contain the actual point information. The aabb argument represents a bounding box for the points section.

```
void NotifyCellDestroyed(ulong cellId);
```

The client notifies the application a given scene cell is no longer used and can be disposed from memory.

```
void NewScene(Dictionary<ulong, bool[]> nextSceneData);
```

The application is notified a new scene configuration has started.

```
void SwapScene(Dictionary<ulong, bool[]> nextSceneData, VoxelFarmCellCache cellCache);
```

The application is notified the new scene is complete and has replaced the previous scene configuration.

```
void Clear();
```

The client asks the application to clear all resources.

```
void Begin();
```

The application is notified new viewing conditions have started (usually as a result of setting new View Metadata)

Setting view focus

Once the view is configured for rendering, the client may start calling the “SetFocus” function. This would happen every time the user moves the viewing position. The “SetFocus()” function instructs the view object to produce a new representation of the data, where the provided coordinates appear in an adequate level of detail:

```
MyView.SetFocus(x, y, z, 0.0, 0, 3);
```

The first three parameters are the (x, y, z) coordinates of the focal point for the view. The fourth argument is the distance from the observer to this point, in this case the “0.0” makes the system assume the observer is right at the focal position.

All spatial units in this call are in Voxel Farm coordinates, to set the focus region using project coordinates, the caller must transform the coordinates first into Voxel Farm space by using the affine transform object that is already initialized in the view object. This transform is held in the “AffineTransform” property.

The fifth argument can be either zero or one, where a value of one increases the view resolution by a factor of 2. The last argument is the radius of the first level of detail.

Per-frame work

At each frame the, the application may call the “Update()” function to the view:

```
MyView.Update();
```

By calling Update, the application asks the view to discover if a new scene configuration is necessary and whether new portions of the data need to be requested.

In most application frameworks, like Unity3D, Unreal Engine 4 and OpenGL, certain operation involving hardware resources must all execute in the same thread. This is usually the main application thread or a rendering thread.

For this reason, an application using the C# client will also be responsible for making calls to “VoxelFarmPool.RunNextMainThreadJob()” from a thread the application considers to be the one in charge of managing resources:

```
VoxelFarmPool.RunNextMainThreadJob();
```

In a system that performs visualization, this function would typically be called every frame. The Voxel Farm client will attempt to minimize the time spent in these calls but since callbacks to the IVoxelFarmRenderer interface will be happen in this thread, it is up to the application to provide the best performance possible.

Example of IVoxelFarmRenderer

The following code listing shows a complete console application that uses the C# Client and defines the IVoxelFarmRenderer interface for callbacks:

```
using System;
using System.Collections.Generic;
using VoxelFarm;
namespace ConsoleApp1
{
   class MyDummyResource : VoxelFarm.IVoxelFarmCellResource
   {
      private string Name;
      private ulong CellId;
      public MyDummyResource(string name, ulong cellId)
      {
         Name = name;
         CellId = cellId;
      }
      public void Release()
      {
         Console.WriteLine("Released " + Name + " for " + CellId);
      }
   }
   class MyRenderer : VoxelFarm.IVoxelFarmRenderer
   {
      public IVoxelFarmCellResource BakeCellMesh(ushort clusterId, ulong cellId, byte layer, byte submesh, uint[]
faces, float[] vertices, float[] normals, byte[] colors)
      {
         Console.WriteLine("Received mesh " + submesh + " for " + cellId);
         return new MyDummyResource("Mesh", cellId);
      }
      public IVoxelFarmCellResource BakeCellPoints(ushort clusterId, ulong cellId, byte layer, float[]
vertices, byte[] colors, double[,] aabb)
      {
         Console.WriteLine("Received points for " + cellId);
```

```
            return new MyDummyResource("Points", cellId);
        }
        public IVoxelFarmCellResource
BakeCellTexture(ushort clusterId, ulong cellId, byte layer, byte textureType, ushort textureSize, byte[]
textureData, byte textureFormat)
        {
            Console.WriteLine("Received texture for " + cellId);
            return new MyDummyResource("Texture", cellId);
        }
        public void Begin()
        {
            Console.WriteLine("New view");
            if (OnNewView != null)
                OnNewView();
        }
        public void Clear()
        {
            Console.WriteLine("Scene cleared");
        }
        public void NewScene(Dictionary<ulong, bool[]> nextSceneData)
        {
            Console.WriteLine("New Scene " + nextSceneData.Count + " cells");
        }
        public void NotifyCellDestroyed(ulong cellId)
        {
            Console.WriteLine("Cell " + cellId + " deleted");
        }
        public void SetVoxelFarmOrigin(double x, double y, double z)
        {
            Console.WriteLine("Origin set to: " + x + ", " + y + ", " + z);
            OriginX = x;
            OriginY = y;
            OriginZ = z;
        }
        public void SwapScene(Dictionary<ulong, bool[]> nextSceneData, VoxelFarmCellCache cellCache)
        {
            Console.WriteLine("Scene completed");
            if (OnSceneCompleted != null)
                OnSceneCompleted();
        }
        public Action OnNewView = null;
```

```
    public Action OnSceneCompleted = null;
    public double OriginX = 0.0;
    public double OriginY = 0.0;
    public double OriginZ = 0.0;
}
class Program
{
    static void Main(string[] args)
    {
        Random rnd = new Random();
        VoxelFarm.VoxelFarmWorkPool.Start(4);
        VoxelFarm.VoxelFarmView vf = new VoxelFarm.VoxelFarmView();
        MyRenderer myRenderer = new MyRenderer();
        myRenderer.OnNewView = () =>
        {
            // we will go to the origin
            vf.SetFocus(
                myRenderer.OriginX,
                myRenderer.OriginY,
                myRenderer.OriginZ,
                0.0, 0, 3);
        };
        myRenderer.OnSceneCompleted = () =>
        {
            // we will go to a new random location
            vf.SetFocus(
                myRenderer.OriginX + 10000.0 * rnd.NextDouble(),
                myRenderer.OriginY + 10000.0 * rnd.NextDouble(),
                myRenderer.OriginZ + 10000.0 * rnd.NextDouble(),
                0.0, 0, 3);
        };
        vf.SetRenderer(myRenderer);
        vf.SetServer("http://localhost", “localhost", 3333, false);
        vf.SetProjectId("sonomadb");
        vf.LoadProject((code) =>
        {
            if (code == System.Net.HttpStatusCode.OK)
            {
                var views = vf.GetEntities(VoxelFarm.VoxelFarmEntityType.View);
                if (views.Count > 0)
                {
```

```
                    vf.SelectView(views[0]["ID"]);
                }
            }
        });
        bool terminate = false;
        while (!terminate)
        {
            VoxelFarm.VoxelFarmWorkPool.RunNextMainThreadJob();
            vf.Update();
        }
    }
}
```

Revision #1
Created 17 March 2025 20:20:01 by admin
Updated 17 March 2025 20:28:54 by admin