Example - Mesh Export

The following C# program connects to a Voxel Farm server and exports the terrain surface within a 3D region as a mesh:

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Net;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using VoxelFarm;



namespace ExportMesh
{
   class Program
   {
      private static void Terrain2MeshOBJ(
         string webServer,
         string contentServerAddr,
         int contentServerPort,
         string projectId,
         string entityId,
         Region25D region,
         int lod,
         string outputFile)
      {
         // Create output file
         System.IO.StreamWriter fileOutput = new StreamWriter(outputFile);

         // Local variables to track OBJ output
         bool done = false;
```

```
        int totalFaces = 0;
        int vertexIndexOffset = 1;

        // Start work pool
        VoxelFarmWorkPool.Start(2 * Environment.ProcessorCount);

        // Create view
        VoxelFarmView view = new VoxelFarmView();

        // Print view debug log
        view.OnDebugLog = (message) =>
        {
           //Console.WriteLine(message);
        };

        // Also print any uncaught error
        VoxelFarmWorkPool.OnError = (Exception error) =>
        {
           Console.WriteLine(error.Message + " " + error.StackTrace);
           done = true;
        };

        // Set OnViewStarted event
        view.OnViewStarted = () =>
        {
           try
           {
              // Use the supplied region to create a set of ranges of 10x10x10 cells
              var ranges = region.breakdownRegion(lod, 10, view.AffineTransform);



              // Iterate over each range of cells
              int totalRanges = ranges.Count;
              int completedRanges = 0;
              foreach (var range in ranges)
              {
                 // Output progress message
                 int progress = (100 * completedRanges++) / totalRanges;
                 string message = (totalFaces > 0) ? String.Format("{0:0,0} triangles exported...", totalFaces)
: "Scanning for triangles...";
```

```
            Console.WriteLine(progress + "% " + message);

            // Ask the view to load the cell range
            view.SetCellRange(range);

            // Wait until the view completes processing the range
            view.Join();

            // Check that we still have a connection to the server
            if (view.ConnectionLost() || !view.IsConnected())
            {
                Console.WriteLine("Lost connection to content server.");
                done = true;
            }
        }
        done = true;
    }
    catch (Exception error)
    {
        Console.WriteLine(error.Message + " " + error.StackTrace);
        done = true;
    }
};


// Define a callaback to handle mesh data
view.OnReceiveMeshData = (data) =>
{
    ManualResetEvent outputDone = new ManualResetEvent(false);
    VoxelFarmWorkPool.RunInMainThread(() => {

        int level, xc, yc, zc;
        VoxelFarmCell.Unpack(data.cellId, out level, out xc, out yc, out zc);
        double scale = VoxelFarmConstant.CELL_SIZE * (1 << level);
        double cellOffsetX = scale * xc;
        double cellOffsetY = scale * yc;
        double cellOffsetZ = scale * zc;

        int vertCount = data.vertices.Length / 3;
        int faceCount = data.faces.Length / 3;
```

```
        for (int i = 0; i < vertCount; i++)
        {
            var wc = view.AffineTransform.VF_TO_WC(
                new VoxelFarmAffineTransform.sAffineVector
                {
                    X = data.vertices[3 * i + 0] * scale + cellOffsetX,
                    Y = data.vertices[3 * i + 1] * scale + cellOffsetY,
                    Z = data.vertices[3 * i + 2] * scale + cellOffsetZ
                });

            fileOutput.WriteLine("v " + (-wc.X) + " " + wc.Z + " " + wc.Y);
        }

        for (int i = 0; i < faceCount; i++)
        {
            fileOutput.WriteLine("f " +
                (data.faces[3 * i + 0] + vertexIndexOffset) + " " +
                (data.faces[3 * i + 1] + vertexIndexOffset) + " " +
                (data.faces[3 * i + 2] + vertexIndexOffset));
        }

        vertexIndexOffset += vertCount;
        totalFaces += faceCount;

        outputDone.Set();
    });
    outputDone.WaitOne();
};


// Set server and project

view.SetServer(webServer, contentServerAddr, contentServerPort, false);
view.SetProjectId(projectId);

// Load the project
view.LoadProject((HttpStatusCode httpcode) =>
{
    if (httpcode == HttpStatusCode.OK)
```

```
            {
                // Find entity
                var terrainEntity = view.GetEntity(entityId);
                if (terrainEntity == null || terrainEntity["file_type"] != VoxelFarmEntityType.VoxelTerrain)
                {
                    done = true;
                    return;
                }

                // Create view metadata
                var components = new List<VoxelFarmMetaComponent>();
                var componentMasks = new List<ushort>();
                List<VoxelFarmMetaLayer> layerStack = new List<VoxelFarmMetaLayer>();
                layerStack.Add(new VoxelFarmMetaLayerHM { Id = entityId, Config = terrainEntity["runtime"] });
                var layers = new VoxelFarmMetaLayerStack(layerStack);
                var surface = new VoxelFarmMetaSurfaceOperation { OpId = VoxelFarmMetaID.META_OP_NONE,
OpA = layers };
                components.Add(surface);
                componentMasks.Add(0x02);
                var componentMaterials = new List<ushort>();
                componentMaterials.Add(0xFFFF);
                var metadata = new VoxelFarmViewMeta(components, componentMasks,
componentMaterials, null);

                // Set view metadata, this will start the connection to the streaming server
                view.SetViewMetaData(metadata);
            }
            else
            {
                done = true;
            }
        });

        while (!done)
        {
            VoxelFarmWorkPool.RunNextMainThreadJob();
        }

        fileOutput.Close();
        VoxelFarmWorkPool.Stop();
```

```
            Console.WriteLine("Export complete.");
        }

        static void Main(string[] args)
        {
            if (args.Length != 8)
            {
                Console.WriteLine("Usage ExportMesh.exe <REST URL> <Streaming Server> <Streaming Port>
<Project Id> <Entity Id> <Region> <LOD> <Output File>");
                return;
            }

            try
            {
                Region25D region = new Region25D();
                region.loadFromString(args[5]);
                Terrain2MeshOBJ(args[0], args[1], Convert.ToInt32(args[2]), args[3], args[4], region,
Convert.ToInt32(args[6]), args[7]);
            }
            catch (Exception error)
            {
                Console.WriteLine(error.Message + " " + error.StackTrace);
            }
        }
    }
}
```

The following command line tests this program over the Sta. Barbara Island dataset, requesting a mesh export for the full island at LOD 5 resolution:

ExportMesh.exe http://18.236.158.139 18.236.158.139 3333 blank 390CC9D2B934446DAEE5EFE9F7D64C4F 1,1.0000000000002274,300,-13253468.999999998,3937303,-13248860.999999998,3937303,-13248860.999999998,3932695,-13253468.999999998,3932695 5 f:\scrap\m1.obj

Note that this program requires the “vfcompression.dll” to be in the same folder as the .EXE, or be in one of the folders specified in the system PATH variable.

The program produces the following output to the console window:

0% Scanning for triangles...

4% 20,378 triangles exported...

8% 59,296 triangles exported...

12% 120,883 triangles exported...

16% 174,325 triangles exported...

20% 198,326 triangles exported...

24% 231,524 triangles exported...

28% 304,836 triangles exported...

32% 367,609 triangles exported...

36% 410,992 triangles exported...

40% 423,511 triangles exported...

44% 448,613 triangles exported...

48% 522,385 triangles exported...

52% 548,733 triangles exported...

56% 592,684 triangles exported...

60% 597,296 triangles exported...

64% 631,860 triangles exported...

68% 694,986 triangles exported...

72% 759,434 triangles exported...

76% 780,406 triangles exported...

80% 781,337 triangles exported...

84% 806,390 triangles exported...

88% 848,185 triangles exported...

92% 869,169 triangles exported...

96% 875,037 triangles exported...

Export complete.

The exported OBJ mesh looks like this:

Revision #2
Created 17 March 2025 20:31:06 by admin
Updated 17 March 2025 20:39:47 by admin