Easy Voxels: Marching Cubes

Technical Documentation

1. Introduction

1.1. Project Description

EasyVoxels: Marching Cubes provides a fast, multi-threaded and reliable way to generate Voxel Geometry using **Dual Marching Cubes and Marching Cubes** algorithms.

1.2. Algorithm

The marching cubes (MC) algorithm is a widely used technique for computing triangular mesh Iso Surfaces from discretely sampled volume data over rectilinear lattices.

Dual Marching Cubes tend to eliminate the poorly shaped triangles often present in MC surfaces.

https://www.researchgate.net/publication/4112407_Dual_Marching_Cubes
http://paulbourke.net/geometry/polygonise/

2. Structures

2.1. FVoxelSettings

Holds the generic settings for Marching Cubes algorithm.

Units `FIntVector`	Grid dimensions sizes in each direction.
Resolution `float`	Voxel Size. Large values are generating lower mesh quality.
ISOLevel `float`	The minimum ISOLevel. All voxel coordinates with values >= ISOLevel will contribute to the mesh.
bInverted `bool`	Whether to invert the generated mesh triangles.
ChunkRadius `FIntVector`	Amount of chunks displayed at once in each direction.
bUseSharedPoints `bool`	If set to true, enables use of shared points to reduce vertices amount.
bForceManifold `bool`	Force Manifold mesh generation. Enabling this removes dualism but is used only by the DMC algorithm. **!!!Experimental!!!**
NormalType `EVoxelNormal`	Normal algorithm to be used when generating Mesh Data
bUseFlatShading `bool`	Whether Normals use flat shading.
bUseOriginalAlgorithm `bool`	Use original Marching Cubes algorithm by Paul Bourke as described here: http://paulbourke.net/geometry/polygonise/

2.2. FDensityPoint

Holds the density data for specific {X, Y, Z} coordinates in the voxel grid.

Value `float`	Point density
Color `FLinearColor`	Point color

3. Marching Cubes Blueprint Library Functions

3.1. Get Chunk Slot From Location

Convert location to chunk slot.

Inputs:

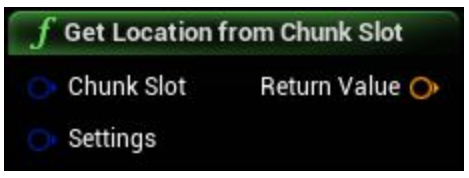
Location `Vector (by ref)`	World Location.
Settings `Voxel Settings Structure (by ref)`	Generic settings for Marching Cubes algorithm.

Outputs:

Return Value `Int Vector Structure`	{X, Y, Z} coordinates of the geometry in the chunk grid.

3.2. Get Location From Chunk Slot

Convert chunk slot to location.



Inputs:

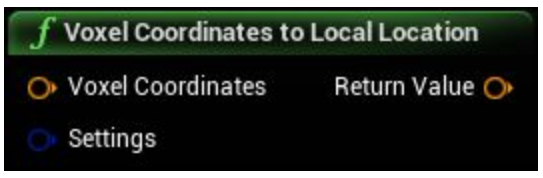
Chunk Slot `Int Vector Structure (by ref)`	{X, Y, Z} coordinates of the geometry in the chunk grid. If not using a chunk system, could use {0, 0, 0}.
Settings `Voxel Settings Structure (by ref)`	Generic settings for Marching Cubes algorithm.

Outputs:

Return Value `Vector`	World Location.

3.3. Voxel CoordinatesTo Local Location

Convert voxel coordinates to Local Space location.



Inputs:

Voxel Coordinates `Vector (by ref)`	{X, Y, Z} coordinates in the voxel grid.
Settings `Voxel Settings Structure (by ref)`	Generic settings for Marching Cubes algorithm.

Outputs:

Return Value `Vector`	Local Space location.

3.4. Get Voxel Coordinates at Location

Get Voxel Coordinates at World Location

Inputs:

Geometry Transform `Transform (by ref)`	Transform of the Geometry/Chunk.
Location `Vector (by ref)`	World Location
Settings `Voxel Settings Structure (by ref)`	Generic settings for Marching Cubes algorithm.

Outputs:

Return Value `Int Vector Structure`	Voxel Coordinates

3.5. Get Voxel Coordinates in Radius

Get Voxel Coordinates in Radius

Inputs:

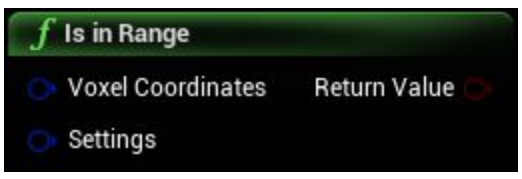
Geometry Transform `Transform (by ref)`	Transform of the Geometry/Chunk.
Brush Location `Vector (by ref)`	Location of the Brush in World Space (Usually Hit Location returned by a Line Trace).
Radius `Vector (by ref)`	Brush Radius
Settings `Voxel Settings Structure (by ref)`	Generic settings for Marching Cubes algorithm.

Outputs:

Return Value `Array of Int Vector Structures`	Array of Voxel Coordinates

3.6. Is in Range

Check if a Voxel Coordinate is in range.



Inputs:

Voxel Coordinates `Int Vector Structure (by ref)`	{X, Y, Z} coordinates in the voxel grid.
Settings `Voxel Settings Structure (by ref)`	Generic settings for Marching Cubes algorithm.

Outputs:

Return Value `Boolean`	Returning false if it goes out of bounds.

3.7. Get Neighbor

Get the neighbor Voxel Coordinate in the given direction.

Inputs:

Voxel Coordinates Int Vector Structure (by ref)	{X, Y, Z} coordinates in the voxel grid.
Direction Int Vector Structure (by ref)	Direction
Settings Voxel Settings Structure (by ref)	Generic settings for Marching Cubes algorithm.

Outputs:

Neighbor Int Vector Structure	Voxel coordinates.
Return Value Boolean	Returning false if it goes out of bounds.

3.8. Get Relevant Slots

Get relevant surrounding slots.

Inputs:

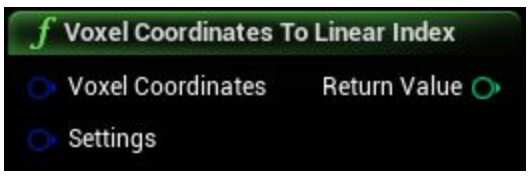
Root Slot Int Vector Structure (by ref)	Root slot to get the surrounding relevant slots for.
Existing Chunks Array of Int Vector Structures	Array of existing slots. This should hold slots that you already spawned. If a relevant slot is contained by ExistingChunks then it will be added to the OldRelevantSlots array.
Settings Voxel Settings Structure (by ref)	Generic settings for Marching Cubes algorithm.

Outputs:

New Slots Array of Int Vector Structures	Array of slots that are relevant and require spawning.
Old Relevant Slots Array of Int Vector Structures	Array of relevant slots that are relevant but you already spawned.

3.9. Voxel Coordinates To Linear Index

Convert Voxel Coordinates to a Linear Index
(This can be used to treat 1D Arrays as 3D)



Inputs:

Voxel Coordinates `Int Vector Structure (by ref)`	{X, Y, Z} coordinates in the voxel grid.
Settings `Voxel Settings Structure (by ref)`	Generic settings for Marching Cubes algorithm.

Outputs:

Return Value `Integer`	Linear Index

3.10. Linear Index To Voxel Coordinates

Convert a Linear Index to Voxel Coordinates
(This can be used to treat 1D Arrays as 3D)

Inputs:

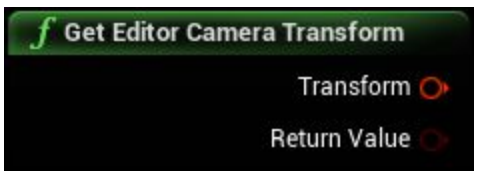
Linear Index `Integer (by ref)`	Linear Index
Settings `Voxel Settings Structure (by ref)`	Generic settings for Marching Cubes algorithm.

Outputs:

Voxel Coordinates `Int Vector Structure`	{X, Y, Z} coordinates in the voxel grid.

3.11. Get Editor Camera Transform

Get Camera Transform in Simulation only.

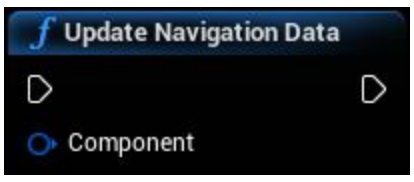


Outputs:

Transform Transform	Camera transform.
Return Value Boolean	Returning false if the camera transform couldn't be retrieved.

3.12. Update Navigation Data

Update Navigation Data



Inputs:

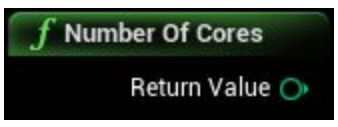
In Exec	
Component Actor Component Object Reference	The component to update navigation data for.

Outputs:

Out Exec	

3.13. Number Of Cores

Return the number of CPU cores

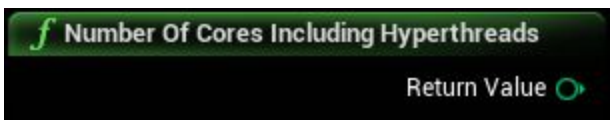


Outputs:

Return Value Integer	Return the number of CPU cores

3.14. Number Of Cores Including Hyperthreads

Return the number of CPU cores including Hyperthreads



Outputs:

Return Value Integer	Return the number of CPU cores including Hyperthreads

4. Density Data Builder

4.1. Introduction

UMarchingCubesDensityBuilder is an abstract class that it's meant to be inherited in order to allow us to provide custom data to the C++ Marching Cubes algorithm at runtime or in the editor. This makes it easy for advanced users as well as beginners to use Easy Voxels.

4.2. Construct Density Point

Dynamically build density data using blueprints. This implementation is used by Marching Cubes algorithm to generate density data.
This function executes on background thread. To keep it secure and prevent possible race condition or other threading issues it is a const function which doesn't allow modification of outside variables.

Inputs:

Target Density Builder Object Reference	
Voxel Coordinates Int Vector Structure (by ref)	{X, Y, Z} coordinates in the voxel grid.
Chunk Slot Int Vector Structure (by ref)	{X, Y, Z} coordinates of the geometry in the chunk grid. If not using a chunk system, could use {0, 0, 0}.
Settings Voxel Settings Structure (by ref)	Generic settings for Marching Cubes algorithm.

Outputs:

Return Value Density Point Structure	Computed density point.

4.3. On Init

This function gets executed on the game thread once the Density Data Builder has been created.

Inputs:

Owning Component Object Reference	Owner Object for generated data.

5. Marching Cubes Component

5.1. Introduction

UMarchingCubesComponent provides an implementation of the engine's PMC. It allows smooth and automatic rendering of Voxel Data.

5.2. Automatic LOD

UMarchingCubicComponent provides an automatic LOD system.

Tooltips explain further.

5.3. Get Current LOD

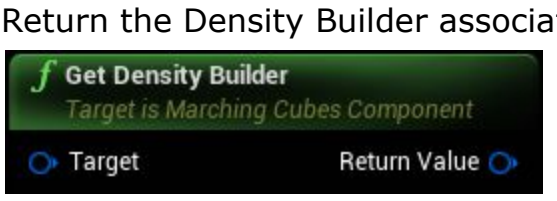
Get the current LOD

Outputs:

Return Value `Integer`	Get the current LOD

5.4. Get Density Builder

Return the Density Builder associated with this component

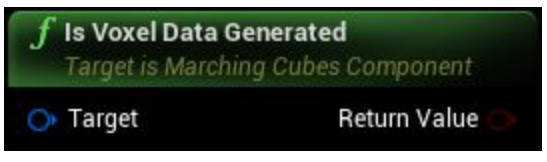


Outputs:

Return Value `Marching Cubes Density Builder Object Reference`	Return the Density Builder associated with this component

5.5. Is Voxel Data Generated

Check if the Voxel Data is marked as generated



Outputs:

Return Value `Boolean`	Check if the Voxel Data is marked as generated

5.6. Request Mesh LOD

Manually Queue a request for LOD generation in the next UpdateInterval iteration

Inputs:

LOD Integer	LOD to be requested

5.7. Request Voxel Data Update

Manually Queue a request for Voxel Data update in the next UpdateInterval iteration

5.8. Get Triangle

Retrieve a Triangle from the current Mesh Section.

Inputs:

Triangle ID Integer	Triangle ID to retrieve

Outputs:

Vertex 1 Proc Mesh Vertex Structure	First Vertex in triangle
Vertex 2 Proc Mesh Vertex Structure	Second Vertex in triangle
Vertex 3 Proc Mesh Vertex Structure	Third Vertex in triangle
Return Value Boolean	Returns true if triangle successfully retrieved

5.9. Get Triangle Count

Retrieve the Triangle Count in the current Mesh Section.

Outputs:

Return Value Integer	Triangle count

5.10. Invalidate Task

Invalidate any running task if any.

5.11. Get Relevant LOD

Computes the current relevant LOD based on distance.

Outputs:

Return Value Integer	Computed LOD

6. Delegates

6.1. OnPreVoxelDataGenerated

This Delegate gets called right before the component begins to generate Voxel Data.

6.2. OnVoxelDataGenerated

This Delegate gets called after RequestMeshLOD generated Mesh Data.
Binding this delegate will overlap internal LOD Request, enabling users to manually call RequestMeshLOD if successful.

6.3. OnPreLODGenerate

This Delegate gets called right before the component begins to generate Mesh Data.

6.4. OnLODGenerated

This Delegate gets called after RequestMeshLOD generated Mesh Data.
Binding this delegate will overlap internal Mesh Section creation/update, enabling users to manually call SetMeshData.
This delegate gets called only when there is valid Mesh Data generated and the RequestMeshLOD task wasn't canceled.

6.5. OnLODUpdate

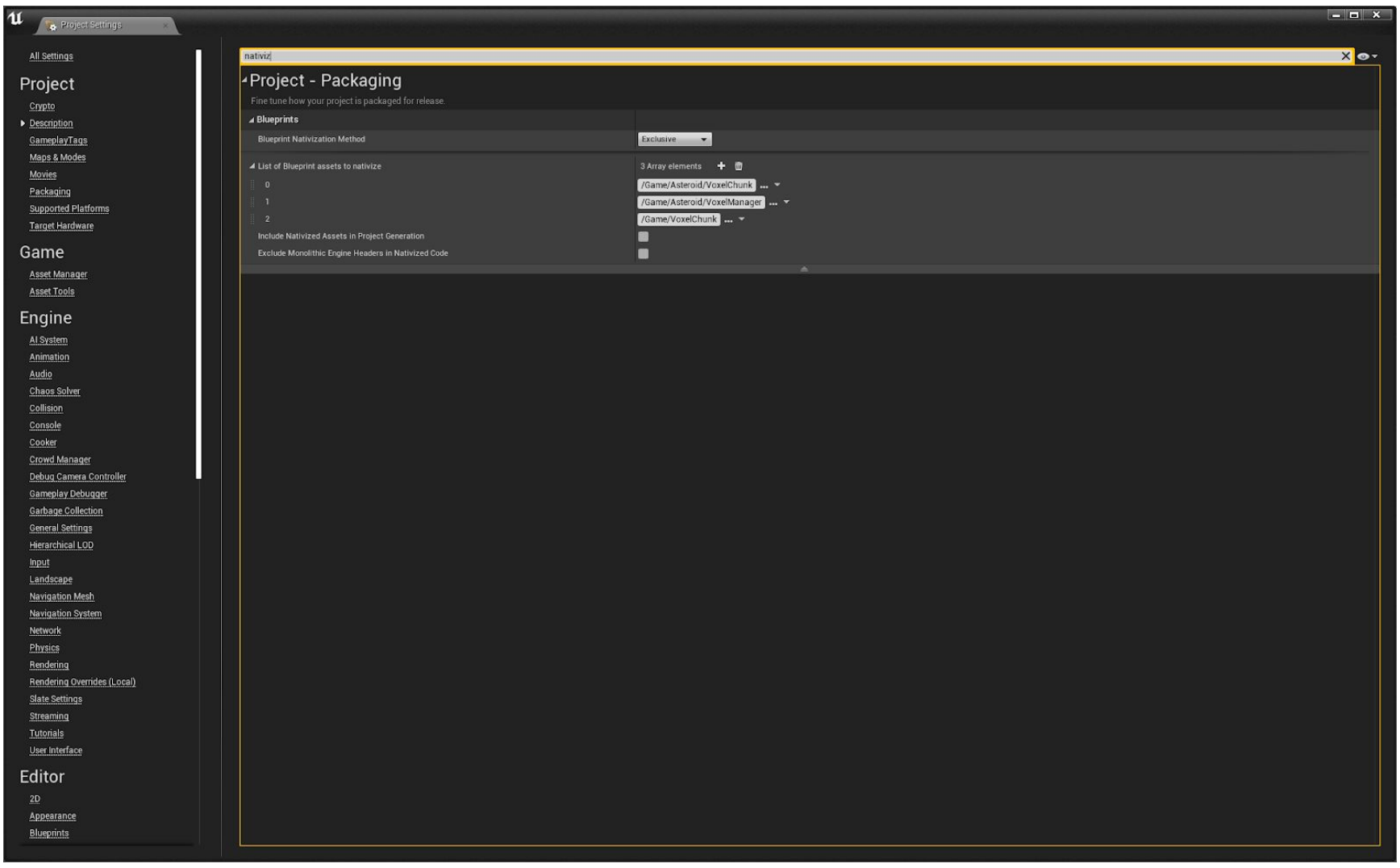
This Delegate gets called after RequestMeshLOD generated Mesh Data which it's just an update and bTryMeshUpdate is enabled.
Binding this delegate will overlap internal Mesh Section creation/update, enabling users to manually call UpdateMeshSection.
This delegate gets called only when there is valid Mesh Data generated and the RequestMeshLOD task wasn't canceled.

6.6. OnLODFail

This Delegate gets called if RequestMeshLOD was canceled or it generated no Mesh Data.
If bNoMeshData is false then the Task was canceled or failed for unknown reasons.

7. Performance

For Packaged games you can have as many chunks/geometry as you need but you should enable nativization for blueprints inheriting **Density Data Builder** Object. This allows for blueprints to be converted to C++ skipping completely Blueprint VM.



8. Conclusions

Overall, Easy Voxels provides a super fast solution for runtime geometry generation.
You can generate terrains, asteroids or any procedural geometry.
Literally **everything**.
Easy Voxels is the result of over 1 and half years of research and has gone through a lot of changes in order to reach its current state.

The speed, robustness and customization abilities are what makes it a very powerful solution.

For more informations:
support@yakistudios.com
https://discord.gg/v4D42Vp

THANK YOU