osuve Documentation

Release 0.0.1

Raus

Dec 19, 2021

Documentation

OSUVE is my take on a voxel engine, made open-source to the world to put so that others may benefit from the knowledge that is accumulated here, and perhaps contribute some of their own. I will be continually working on this until it becomes a working product. (After all, I have plans to make a game with it!)

Be sure to join the official Discord!

CHAPTER 1

World

class World

GameObject **_chunkPrefab** { }

Dictionary<*ChunkPos,* DataChunk> **_chunks** { }

Dictionary<*ColumnPos,* DataColumn> **_columns** { }

Dictionary<*ChunkPos,* DataChunk> **_offloacChunks** { }

SimplePriorityQueue<*Chunk*> **_loadQueue** { }

Boolean **_rendering** { }

static Int32 **_viewRangeHorizontal** { }

static Int32 **_viewRangeVertical** { }

static *ChunkPos* **_playerPos** { }

public Int32 **chunkSize** { **get;** }
The size (width, breadth, and height) of chunks.

public static *Atlas.ID* **GetBlock** (*BlockPos pos*)
Fetches a block that has already been generated in the past.

public static *Atlas.ID* **GenerateBlock** (*BlockPos pos*)
Generates a block for a given position.

public static Int32 **GenerateTopology** (Int32 *x*, Int32 *z*)
Generates the topological height of the stone layer for a given coordinate.

public static Single **SimplexNoise** (Single *x*, Single *y*, Single *z*, Single *scale*, Single *height*, Single *power*)
Returns simplex noise from a given point, modulated by some variables.

public static Int32 **GetViewRange** ()
Gets view range.

public static *DataChunk* **GetChunk** (*ChunkPos* *pos*)
Gets the `DataChunk` from the given pos.

public static *DataColumn* **GetColumn** (*ColumnPos* *pos*)
Gets the `DataColumn` from the given pos.

void **RenderThread** ()
Special function that does threaded generation.

void **GenerateChunks** ()
Generates all possible chunk positions that are in view range if a chunk does not already exist at that position.

void **PingChunks** ()
Checks whether chunks are still in view range or not, and destrys them if need be.

void **DestroyChunk** (*ChunkPos* *pos*)
Safely removes a `Chunk` from the chunk dictionary.

CHAPTER 2

Chunk

class Chunk

List<Vector3> **_newVerts** { }

List<Int32> **_newTris** { }

List<Vector2> **_newUV** { }

List<Color> **_newColors** { }

Int32 **_faceCount** { }

Mesh **_mesh** { }

MeshCollider **_col** { }

Boolean **_updateMesh** { }

Boolean **_clearMesh** { }

State **_state** { }

Int32 **_chunkSize** { }

ChunkPos **_chunkPos** { }

DataChunk **_chunkData** { }

public void **LoadData** (*ChunkPos pos*, *DataChunk chunkData*)
Loads data into chunk.

public void **UpdateState** ()
Lets chunk know it can properly update its state.

public void **GenerateBlocks** ()
Tells chunk to generate its blocks.

public void **GenerateMesh** ()
Tells chunk to generate its mesh.

Atlas.ID **Block** (*BlockPos pos*)
Get block from position

UpdateMesh ()
Updates mesh.

void **CubeUp** (Int32 *x*, Int32 *y*, Int32 *z*, *Atlas.ID block*)

void **CubeDown** (Int32 *x*, Int32 *y*, Int32 *z*, *Atlas.ID block*)

void **CubeNorth** (Int32 *x*, Int32 *y*, Int32 *z*, *Atlas.ID block*)

void **CubeSouth** (Int32 *x*, Int32 *y*, Int32 *z*, *Atlas.ID block*)

void **CubeEast** (Int32 *x*, Int32 *y*, Int32 *z*, *Atlas.ID block*)

void **CubeWest** (Int32 *x*, Int32 *y*, Int32 *z*, *Atlas.ID block*)

void **Cube** (Vector2 *texturePos*)

enum State

Fresh
The chunk has been freshly created, but it has no data associated.

Prepped
The chunk now has its basic data.

Generating
The chunk is actively generating or retrieving block data.

Loaded
The chunk has its block data loaded.

Rendered
The chunk is actively rendering.

CHAPTER 3

Atlas

class Atlas

static Single **tUnit** { **get;** }
Unit length of a cube in the texture atlas.

public Vector2 **GetTexture** (*Atlas.ID dir*, *Atlas.Dir dir*)
Gets the texture UV coordinate for a given block ID.

enum ID

Air

Solid
A special block that represents a generic solid.

Stone

Grass

Dirt

Coal

Log

Leaves

enum Dir

Up

Down

North

South

East

West

CHAPTER 4

DataChunk

class DataChunk

ChunkPos **_pos** { }

Chunk **_chunk** { }

Atlas.ID[] **_blocks** { }

DataColumn **_column** { }

Boolean **_generated** { }

Int32 **_density** { }

DataChunk (*ChunkPos pos*, *Chunk chunk*, *DataColumn column*)
Create a new `DataChunk`.

public void **GenerateBlocks** ()
Tell `DataChunk` to generate its blocks.

public void **SetBlock** (*Atlas.ID block*, *BlockPos pos*)
Set block at given position.

public void **RemoveBlock** (*BlockPos pos*)
Remove block at given position.

public Atlas.ID **GetBlock** (*BlockPos pos*)
Get block at given position.

public void **SetChunk** (*Chunk chunk*)
Assign `Chunk` script.

public Chunk **GetChunk** ()
Get `Chunk` script.

public DataColumn **GetColumn** ()
Get column data.

public Boolean **IsGenerated** ()
Checks whether the data chunk's blocks are generated.

public Boolean **IsEmpty** ()
Checks whether chunk is empty.

CHAPTER 5

DataColumn

class DataColumn

ColumnPos **_pos** { }

Int32[] **_surface** { }

DataColumn (*ColumnPos pos*)
Create a new *DataColumn*.

public Int32 **GetSurface** (Int32 *x*, Int32 *z*)
Gets surface height from given position.

CHAPTER 6

ChunkPos

class ChunkPos

Int32 **x** { **get; set;** }
X coordinate.

Int32 **y** { **get; set;** }
Y coordinate.

Int32 **z** { **get; set;** }
Z coordinate.

static ChunkPos **zero** { **get;** }
Shorthand for writing ChunkPos(0, 0, 0).

static ChunkPos **up** { **get;** }
Shorthand for writing ChunkPos(0, 1, 0).

static ChunkPos **down** { **get;** }
Shorthand for writing ChunkPos(0, -1, 0).

static ChunkPos **north** { **get;** }
Shorthand for writing ChunkPos(0, 0, 1).

static ChunkPos **south** { **get;** }
Shorthand for writing ChunkPos(0, 0, -1).

static ChunkPos **east** { **get;** }
Shorthand for writing ChunkPos(1, 0, 0).

static ChunkPos **west** { **get;** }
Shorthand for writing ChunkPos(-1, 0, 0).

ChunkPos（Int32 *x*, Int32 *y*, Int32 *z*）
Creates a new `ChunkPos` using explicit x, y, and z coordinates.

ChunkPos（Vector3 *vec*）
Creates a new `ChunkPos` using a Vector3.

public static Single **Distance** (*ChunkPos one*, *ChunkPos two*)
Returns the distance between two `ChunkPos`.

public static Single **CubeDistance** (*ChunkPos one*, *ChunkPos two*)
Returns the cubic distance between two `ChunkPos`.

CHAPTER 7

ColumnPos

class ColumnPos

Int32 **x** { **get; set;** }
X coordinate.

Int32 **z** { **get; set;** }
Z coordinate.

static ColumnPos **zero** { **get;** }
Shorthand for writing ColumnPos(0, 0).

static ColumnPos **north** { **get;** }
Shorthand for writing ColumnPos(0, 1).

static ColumnPos **south** { **get;** }
Shorthand for writing ColumnPos(0, -1).

static ColumnPos **east** { **get;** }
Shorthand for writing ColumnPos(1, 0).

static ColumnPos **west** { **get;** }
Shorthand for writing ColumnPos(-1, 0).

ColumnPos (Int32 *x*, Int32 *z*)
Create a new `ColumnPos` using explicit x and z coordinates.

CHAPTER 8

BlockPos

class BlockPos

Int32 **x** { **get; set;** }
Local x coordinate.

Int32 **y** { **get; set;** }
local y coordinate.

Int32 **z** { **get; set;** }
local z coordinate.

ChunkPos **chunkPos** { **get; set;** }
The chunk this block is located in.

static BlockPos **zero** { **get;** }
Shorthand for writing BlockPos(0, 0, 0).

static BlockPos **up** { **get;** }
Shorthand for writing BlockPos(0, 1, 0).

static BlockPos **down** { **get;** }
Shorthand for writing BlockPos(0, -1, 0).

static BlockPos **north** { **get;** }
Shorthand for writing BlockPos(0, 0, 1).

static BlockPos **south** { **get;** }
Shorthand for writing BlockPos(0, 0, -1).

static BlockPos **east** { **get;** }
Shorthand for writing BlockPos(1, 0, 0).

static BlockPos **west** { **get;** }
Shorthand for writing BlockPos(-1, 0, 0).

BlockPos (Int32 *x*, Int32 *y*, Int32 *z*, *ChunkPos chunkPos = ChunkPos.zero*)
Creates a new `BlockPos` using explicit x, y, and z coordinates inside a ChunkPos.

public Int32 **GetWorldX** ()
Returns global x coordinate.

public Int32 **GetWorldY** ()
Returns global y coordinate.

public Int32 **GetWorldZ** ()
Returns global z coordinate.

Int32 **Mod** (Int32 *x*, Int32 *m*)
The default C# modulus operator % returns negative numbers for negative input, which is undesirable.

void **Correct** ()
This corrects overflown coordinates, appropriately shifting chunkPos and clamping x, y, and z.