THREE.BufferGeometry Serialize to Bobj

Author:Hou,Chunlei

|  |  |  |  |
| --- | --- | --- | --- |
| 2 Byte | Uint32 | meshCount - Mesh Counts | meshPart |
|  |  |
|  |  |
|  |  |
| 2Byte | Uint32 | nameLength - Mesh name length |
|  |  |
|  |  |
|  |  |
| 2 X nameLength Byte | Uint8  .  .  .  Uint8 | nameStrBlock - Mesh Name Title |
| (2 - nameLength % 2) Byte | Uint32  .  Uint32 | [null] or x |
| 2 Byte | Uint32 | lengthOfPositions |
|  |  |
|  |  |
|  |  |
| 2 Byte | Uint16 | hasNormalBlock |
| 2 Byte | Uint16 | [null] |
| 2 Byte | Uint16 | hasUvBlock |
| 2 Byte | Uint16 | [null] |
| 3 X 2 X lengthOfPositions Byte | Float32  .  .  .  Float32 | Vertex position |
| 3 X 2 X lengthOfPositions Byte | Float32  .  .  .  Float32 | If (hasNormalBlock == 1) => normals |
| 2 X 2 X lengthOfPositions Byte | Float32  .  .  .  Float32 | If (hasUvBlock == 1) => UVs |
| meshCount \* meshPart | | | meshPart |

/\*\*

\* [parse description]

\* @method parse

\* @param {ArrayBuffer} buffer [description]

\* @return {THREE.Object3D} [description]

\*/

public parse(buffer:ArrayBuffer):THREE.Object3D{

let result:THREE.Object3D = new THREE.Object3D;

for (var dataView = new DataView(buffer), byteOffset = 0, meshCount = dataView.getUint32(byteOffset, !0), byteOffset = byteOffset + 4, iterator = 0; iterator < meshCount; ++iterator) {

var nameLength = dataView.getUint32(byteOffset, !0)

, byteOffset = byteOffset + 4

, nameStrBlock = new Uint8Array(buffer,byteOffset,nameLength)

, strUintArray = String.fromCharCode.apply(null , nameStrBlock)

, byteOffset = byteOffset + nameLength

, byteOffset = byteOffset + (4 - nameLength % 4)

, lengthOfPositions = dataView.getUint32(byteOffset, !0)

, byteOffset = byteOffset + 4

, hasNormalBlock = dataView.getUint16(byteOffset, !0)

, byteOffset = byteOffset + 2

, hasUvBlock = dataView.getUint16(byteOffset, !0)

, byteOffset = byteOffset + 2

, geometryPart = new THREE.BufferGeometry;

geometryPart.addAttribute("position", new THREE.BufferAttribute(new Float32Array(buffer,byteOffset,3 \* lengthOfPositions),3));

byteOffset += 12 \* lengthOfPositions;

geometryPart.vertices = geometryPart.attributes.position.array;

hasNormalBlock ? (geometryPart.addAttribute("normal", new THREE.BufferAttribute(new Float32Array(buffer,byteOffset,3 \* lengthOfPositions),3)), byteOffset += 12 \* lengthOfPositions, geometryPart.normals = geometryPart.attributes.normal.array) : geometryPart.computeVertexNormals();

hasUvBlock && (geometryPart.addAttribute("uv", new THREE.BufferAttribute(new Float32Array(buffer,byteOffset,2 \* lengthOfPositions),2)), byteOffset += 8 \* lengthOfPositions, geometryPart.uvs = geometryPart.attributes.uv.array);

var meshPart = new THREE.Mesh(geometryPart,this.\_getDefaultMaterial());

meshPart.name = strUintArray;

result.add(meshPart);

}

return result;

}