BufferGeometry

A representation of mesh, line, or point geometry. Includes vertex positions, face indices, normals, colors, UVs, and custom attributes within buffers, reducing the cost of passing all this data to the GPU.

To read and edit data in BufferGeometry attributes, see BufferAttribute documentation.

Code Example

```
const geometry = new THREE.BufferGeometry();

// create a simple square shape. We duplicate the top left and bottom
right
// vertices because each vertex needs to appear once per triangle.
const vertices = new Float32Array( [
        -1.0, -1.0,  1.0, // v0
         1.0, -1.0,  1.0, // v1
         1.0,  1.0,  1.0, // v2

         1.0,  1.0,  1.0, // v3
        -1.0,  1.0,  1.0, // v4
        -1.0, -1.0,  1.0  // v5
] );

// itemSize = 3 because there are 3 values (components) per vertex
geometry.setAttribute( 'position', new THREE.BufferAttribute( vertices,
3 ) );
const material = new THREE.MeshBasicMaterial( { color: 0xff0000 } );
const mesh = new THREE.Mesh( geometry, material );
```

Code Example (Index)

```
const geometry = new THREE.BufferGeometry();

const vertices = new Float32Array( [
        -1.0, -1.0,  1.0, // v0
         1.0, -1.0,  1.0, // v1
         1.0,  1.0,  1.0, // v2
        -1.0,  1.0,  1.0, // v3
] );

const indices = [
        0, 1, 2,
        2, 3, 0,
];

geometry.setIndex( indices );
geometry.setAttribute( 'position', new THREE.BufferAttribute( vertices,
3 ) );

const material = new THREE.MeshBasicMaterial( { color: 0xff0000 } );
const mesh = new THREE.Mesh( geometry, material );
```

Examples

Mesh with non-indexed faces
Mesh with indexed faces
Lines
Indexed Lines
Particles
Raw Shaders

Constructor

BufferGeometry()

This creates a new BufferGeometry. It also sets several properties to a default value.

Properties

.attributes : Object

This hashmap has as id the name of the attribute to be set and as value the buffer to set it to. Rather than accessing this property directly, use .setAttribute and .getAttribute to access attributes of this geometry.

.boundingBox : Box3

Bounding box for the bufferGeometry, which can be calculated with .computeBoundingBox(). Default is `null`.

.boundingSphere : Sphere

Bounding sphere for the bufferGeometry, which can be calculated with .computeBoundingSphere(). Default is `null`.

.drawRange : Object

Determines the part of the geometry to render. This should not be set directly, instead use .setDrawRange. Default is
`{ start: 0, count: Infinity }`
For non-indexed BufferGeometry, count is the number of vertices to render. For indexed BufferGeometry, count is the number of indices to render.

.groups : Array

Split the geometry into groups, each of which will be rendered in a separate WebGL draw call. This allows an array of materials to be used with the geometry.

Each group is an object of the form:
`{ start: Integer, count: Integer, materialIndex: Integer }`
where start specifies the first element in this draw call – the first vertex for non-indexed geometry, otherwise the first triangle index. Count specifies how many vertices (or indices) are included, and materialIndex specifies the material array index to use.

Use .addGroup to add groups, rather than modifying this array directly.

Every vertex and index must belong to exactly one group — groups must not share vertices or indices, and must not leave vertices or indices unused.

.id : Integer

Unique number for this bufferGeometry instance.

.index : BufferAttribute

Allows for vertices to be re-used across multiple triangles; this is called using "indexed triangles". Each triangle is associated with the indices of three vertices. This attribute therefore stores the index of each vertex for each triangular face. If this attribute is not set, the renderer assumes that each three contiguous positions represent a single triangle. Default is `null`.

.isBufferGeometry : Boolean

Read-only flag to check if a given object is of type BufferGeometry.

.morphAttributes : Object

Hashmap of BufferAttributes holding details of the geometry's morph targets.
Note: Once the geometry has been rendered, the morph attribute data cannot be changed. You will have to call .dispose(), and create a new instance of BufferGeometry.

.morphTargetsRelative : Boolean

Used to control the morph target behavior; when set to true, the morph target data is treated as relative offsets, rather than as absolute positions/normals. Default is `false`.

.name : String

Optional name for this bufferGeometry instance. Default is an empty string.

.userData : Object

An object that can be used to store custom data about the BufferGeometry. It should not hold references to functions as these will not be cloned. Default is an empty object `{}`.

.uuid : String

UUID of this object instance. This gets automatically assigned and shouldn't be edited.

Methods

EventDispatcher methods are available on this class.

.addGroup (start : Integer, count : Integer, materialIndex : Integer) : undefined

Adds a group to this geometry; see the groups property for details.

.applyMatrix4 (matrix : Matrix4) : this

Applies the matrix transform to the geometry.

.applyQuaternion (quaternion : Quaternion) : this

Applies the rotation represented by the quaternion to the geometry.

.center () : this

Center the geometry based on the bounding box.

.clearGroups ( ) : undefined

Clears all groups.

.clone () : BufferGeometry

Creates a clone of this BufferGeometry.

.computeBoundingBox () : undefined

Computes the bounding box of the geometry, and updates the .boundingBox attribute. The bounding box is not computed by the engine; it must be computed by your app. You may need to recompute the bounding box if the geometry vertices are modified.

.computeBoundingSphere () : undefined

Computes the bounding sphere of the geometry, and updates the .boundingSphere attribute. The engine automatically computes the bounding sphere when it is needed, e.g., for ray casting or view frustum culling. You may need to recompute the bounding sphere if the geometry vertices are modified.

.computeTangents () : undefined

Calculates and adds a tangent attribute to this geometry.
The computation is only supported for indexed geometries and if position, normal, and uv attributes are defined. When using a tangent space normal map, prefer the MikkTSpace algorithm provided by BufferGeometryUtils.computeMikkTSpaceTangents instead.

.computeVertexNormals () : undefined

Computes vertex normals for the given vertex data. For indexed geometries, the method sets each vertex normal to be the average of the face normals of the faces that share that vertex. For non-indexed geometries, vertices are not shared, and the method sets each vertex normal to be the same as the face normal.

.copy (bufferGeometry : BufferGeometry) : this

Copies another BufferGeometry to this BufferGeometry.

.deleteAttribute (name : String) : BufferAttribute

Deletes the attribute with the specified name.

.dispose () : undefined

Frees the GPU-related resources allocated by this instance. Call this method whenever this instance is no longer used in your app.

.getAttribute (name : String) : BufferAttribute

Returns the attribute with the specified name.

.getIndex () : BufferAttribute

Return the .index buffer.

.hasAttribute (name : String) : Boolean

Returns `true` if the attribute with the specified name exists.

.lookAt (vector : Vector3) : this

vector - A world vector to look at.

Rotates the geometry to face a point in space. This is typically done as a one time operation, and not during a loop. Use Object3D.lookAt for typical real-time mesh usage.

.normalizeNormals () : undefined

Every normal vector in a geometry will have a magnitude of `1` . This will correct lighting on the geometry surfaces.

.rotateX (radians : Float) : this

Rotate the geometry about the X axis. This is typically done as a one time operation, and not during a loop. Use Object3D.rotation for typical real-time mesh rotation.

.rotateY (radians : Float) : this

Rotate the geometry about the Y axis. This is typically done as a one time operation, and not during a loop. Use Object3D.rotation for typical real-time mesh rotation.

.rotateZ (radians : Float) : this

Rotate the geometry about the Z axis. This is typically done as a one time operation, and not during a loop. Use Object3D.rotation for typical real-time mesh rotation.

.scale (x : Float, y : Float, z : Float) : this

Scale the geometry data. This is typically done as a one time operation, and not during a loop. Use Object3D.scale for typical real-time mesh scaling.

.setAttribute (name : String, attribute : BufferAttribute) : this

Sets an attribute to this geometry. Use this rather than the attributes property, because an internal hashmap of .attributes is maintained to speed up iterating over attributes.

.setDrawRange (start : Integer, count : Integer) : undefined

Set the .drawRange property. For non-indexed BufferGeometry, count is the number of vertices to render. For indexed BufferGeometry, count is the number of indices to render.

.setFromPoints (points : Array) : this

Defines a geometry by creating a `position` attribute based on the given array of points. The array can hold instances of Vector2 or Vector3. When using two-dimensional data, the `z` coordinate for all vertices is set to `0`.
If the method is used with an existing `position` attribute, the vertex data are overwritten with the data from the array. The length of the array must match the vertex count.

.setIndex (index : BufferAttribute) : this

Set the .index buffer.

.toJSON () : Object

Convert the buffer geometry to three.js JSON Object/Scene format.

.toNonIndexed () : BufferGeometry

Return a non-index version of an indexed BufferGeometry.

.translate (x : Float, y : Float, z : Float) : this

Translate the geometry. This is typically done as a one time operation, and not during a loop. Use Object3D.position for typical real-time mesh translation.

Source

src/core/BufferGeometry.js