ASTRUM
FORGE

Executive Summary

This white paper introduces **Bit Voxels**, a memory-efficient subdivision technique for voxel-based game maps, alongside **Geometry Tables** and **Geometry Indexing** to enhance rendering and storage. These approaches tackle key challenges in voxel game maps, offering improved performance and scalability for real-time applications.

## Introduction

Voxel-based games, like *Minecraft* and *No Man's Sky*, create vast, procedurally generated worlds. These environments are composed of volumetric pixels, known as **voxels**, which allow for dynamic, destructible landscapes. However, managing large voxel datasets presents challenges in memory consumption, rendering performance, and real-time updates.

This paper highlights three innovations designed to improve voxel-based maps:

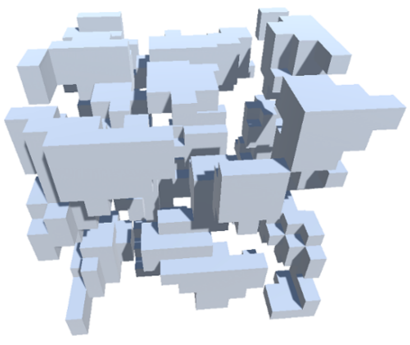
1. **Bit Voxels**: A memory-efficient technique to reduce voxel data storage.
2. **Geometry Tables**: A precomputed lookup system for voxel surface geometry.
3. **Geometry Indexing**: A method to optimize rendering by focusing on visible surface voxels only.

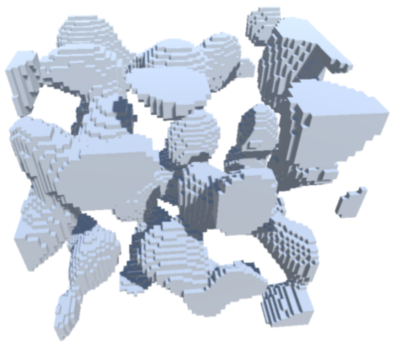
## Bit Voxels: Optimized Subdivision for High-Resolution Maps

Traditional voxel maps are represented as 3D grids, with each voxel storing data like material type and texture. However, as the resolution of these maps increases, the memory required grows exponentially. Bit Voxels offer a solution to this.

### Comparison of Traditional vs. Bit Voxels

To visualize the difference, consider a 1024x1024x1024 voxel map:


Before


After

In a traditional voxel system, each voxel stores 32 bits of data. For a 1024x1024x1024 map, this requires over 4GB of memory. As the map resolution grows, so does the memory consumption, often with redundant storage of internal voxels that are never seen by the player.

**Bit Voxels (BV)** solve this by subdividing each full voxel (FV) into a 4x4x4 grid of smaller voxels, using just one bit per subdivision to indicate visibility. This approach dramatically reduces memory usage—by up to 20x—bringing a 4GB map down to around 200MB, without sacrificing the ability to render high-detail surfaces.

### Memory Efficiency in Real-Time Applications

Bit Voxels shine in environments where large, destructible worlds need to be rendered in real-time. By reducing the memory load, they enable faster load times, better performance on platforms with limited resources (such as mobile devices or VR systems), and higher resolution environments, even as players alter the world.

## Geometry Tables and Geometry Indexing

Rendering all voxels in a 3D map is resource-intensive, especially when many of those voxels are hidden from view. Geometry Tables and Geometry Indexing optimize this process by focusing on what truly matters: the visible parts.

### Rendering Optimization Through Geometry Indexing

In dynamic environments where objects can be altered or destroyed, Geometry Indexing simplifies rendering by identifying which voxel faces are visible. Rather than processing every voxel, only the surfaces visible to the player are rendered, reducing the computational load.

### Before/After Comparison of Rendered Voxels

Before Geometry Indexing, a voxel scene might require rendering hundreds of thousands of individual polygons. For example, a 32x32x32 voxel section could result in over 390,000 triangles needing to be processed.

With Geometry Indexing, this number is significantly reduced—down to approximately 12,000 triangles. The result is a noticeable improvement in real-time applications, especially in interactive voxel-based environments where destruction or construction takes place.

## Efficient Storage for Large Voxel Maps

Scalability is crucial for voxel-based worlds, especially when they are procedurally generated. Bit Voxels, combined with efficient storage techniques, allow these large environments to be managed effectively.

### Morton Keys for Spatial Indexing

Morton Keys (Z-order curves) are employed to map 3D voxel coordinates to 1D indices while preserving spatial locality. By interleaving the x, y, and z coordinates into a single integer, Morton Keys ensure that neighboring voxels remain close in memory, improving cache performance and speeding up access.

### Hash Map Voxel Storage

Voxel chunks are stored in hash maps, each addressed by its Morton Key. This allows for flexible, dynamic expansion of the voxel world as players explore or modify the environment, without requiring large, pre-allocated grids. As a result, the system can grow organically with the game.

## Conclusion

By combining Bit Voxels, Geometry Tables, and Geometry Indexing, this white paper presents a scalable and efficient solution for rendering and managing large, dynamic voxel maps. These techniques allow for higher resolution, real-time rendering, and significantly reduce memory and processing overhead.

Future work will focus on GPU-based implementations and exploring data compression methods to support multi-user environments.

For reference implementation, visit: https://github.com/astrum-forge/bvx-kit.