In the following sections, we first propose a general-purpose GPU-based voxelization algorithm that converts an arbitrary model into a Cartesian grid volume. Regardless, the handling of the boundary conditions requires discrete boundary nodes to be aligned with the LBM lattice. Even if the boundary precludes this alignment, the corresponding color channel is simply updated to the old value.

Equation 4

\[ f_i(x) = \frac{1}{\sqrt{\gamma}} \sum_{j} c \left( \frac{x}{\delta} \right) c \left( \frac{x - e_j}{\delta} \right) \]

For obstacle boundaries, we adopt the improved bounce-back rule of Mei et al. 2000, which can handle deformable and moving boundaries. As indicated in the previous pass. A fragment reaches the frame buffer only if its depth value is greater than that of the corresponding pixel in the depth texture, while the attribute is the 3D position, and the other attributes depend on the application. Assume that the maximum size along any of the major axes of the object is \( n \), and that the value along each axis is \( n + x \).

Instead of using a stack of 2D textures to represent a volume, we tile the slices into a large 2D texture that can be considered a "flat" volume. Similar to Equation 1, only a subset of all possible boundary nodes are active in the final simulation. If the texture width is \( w \), then \( x = \frac{w}{n} \).

The images containing the voxel attributes are then copied to a vertex array allocated in video memory (using OpenGL extensions such as ARB_pixel_buffer_object and ARB_vertex_buffer_object). Each vertex corresponds to a single voxel and contains a position, a color, and a data value associated with that voxel. The position is computed as a linear interpolation of the vertex's coordinates and the coordinates of its neighbors. The color is derived from a 1D texture lookup, where the texture coordinates are obtained by subtracting the position from the vertex's coordinates. The data value is stored in a separate buffer and can be accessed through vertex programs.

Figure 47-4 is a diagram of the data flow of the LBM computation on the GPU; green boxes represent the textures storing lattice properties, and blue boxes are the vertex arrays. The data is copied from the textures to the vertex arrays, which are then processed by the vertex program to generate the final frame buffer.

Figure 47-9 shows the time in milliseconds per step of the 3D LBM model as a function of the lattice size, running on both the CPU and the GPU. Note that both implementations do not utilize SSE.