Vulkanised 2026

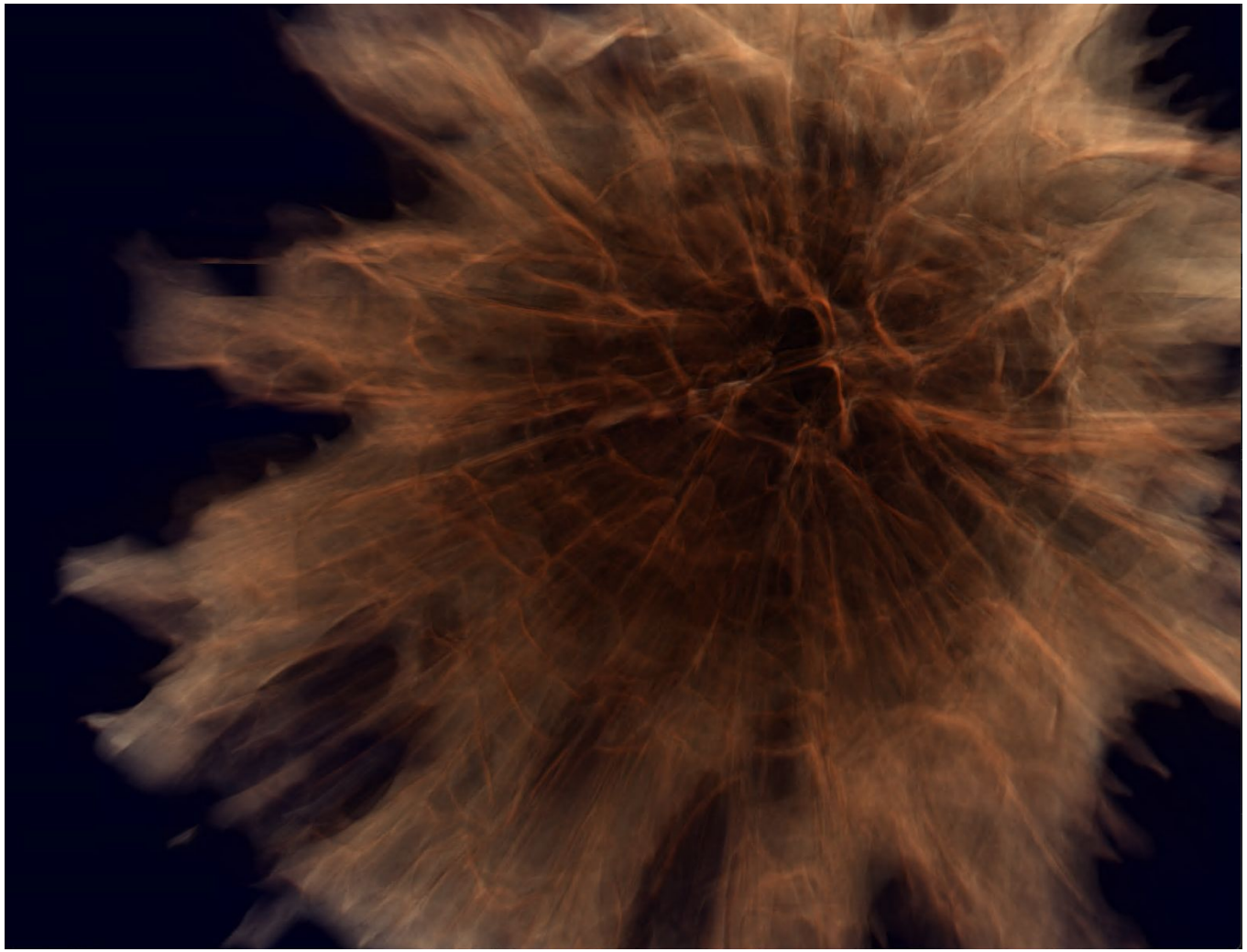
The 8th Vulkan Developer Conference
San Diego, USA | February 9-11, 2026

Digital Corals: Visualizing Nature's Complexity

Mark Bo Jensen, Assistant Professor, Technical
University of Denmark

Outline

- The Corals (the project)
- The Compute (The Engine)
- And The Calls (The API)



CORALS - New strategies for harvesting solar energy using coral-inspired microgeometry

Jeppe Revall Frisvad
Section for Visual Computing
Technical University of Denmark

&

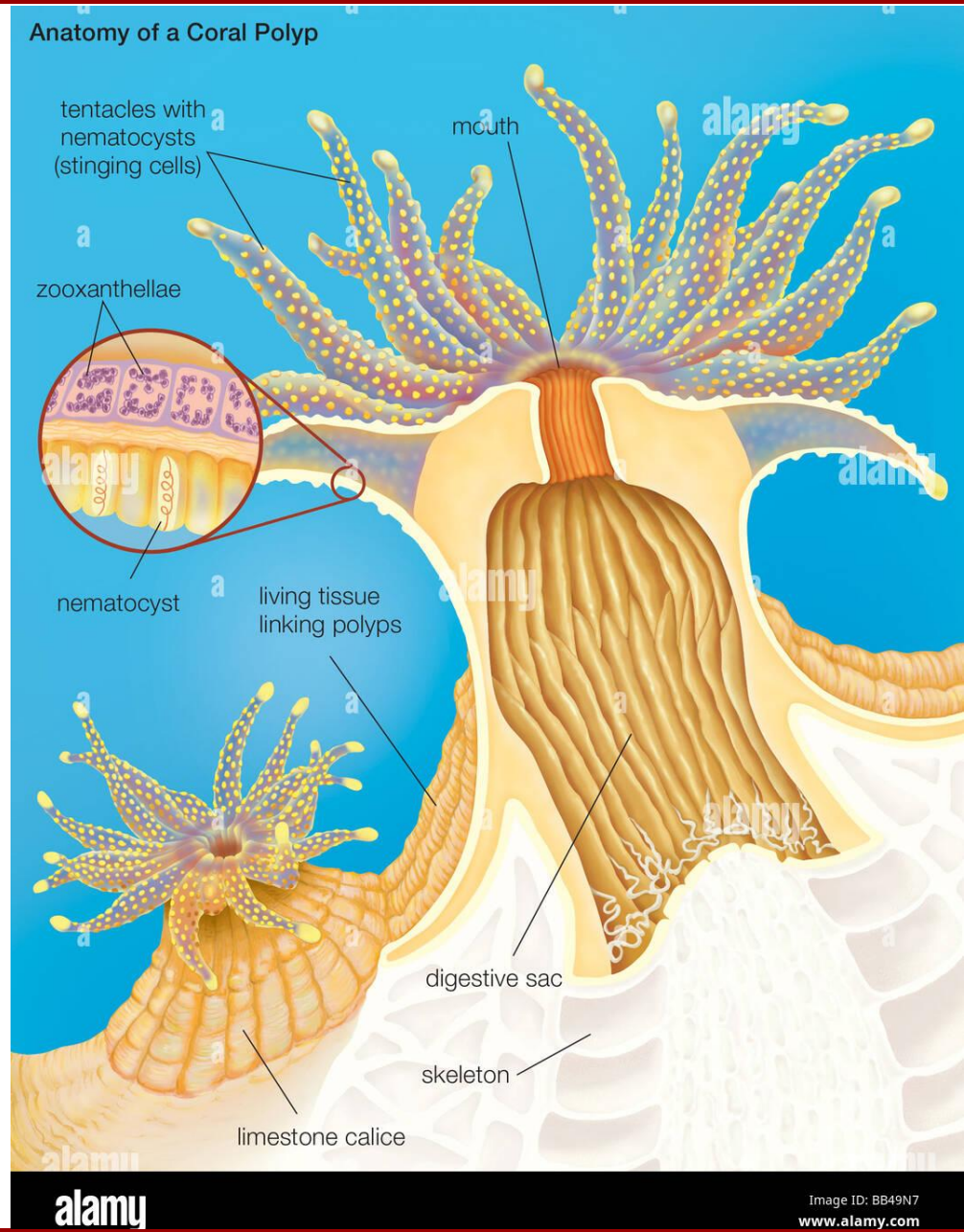
Michael Kühl
Marine Biology Section
University of Copenhagen

With Alina Pranovich, Swathi Murthy, Mark Bo Jensen

VILLUM FONDEN

Why Corals?

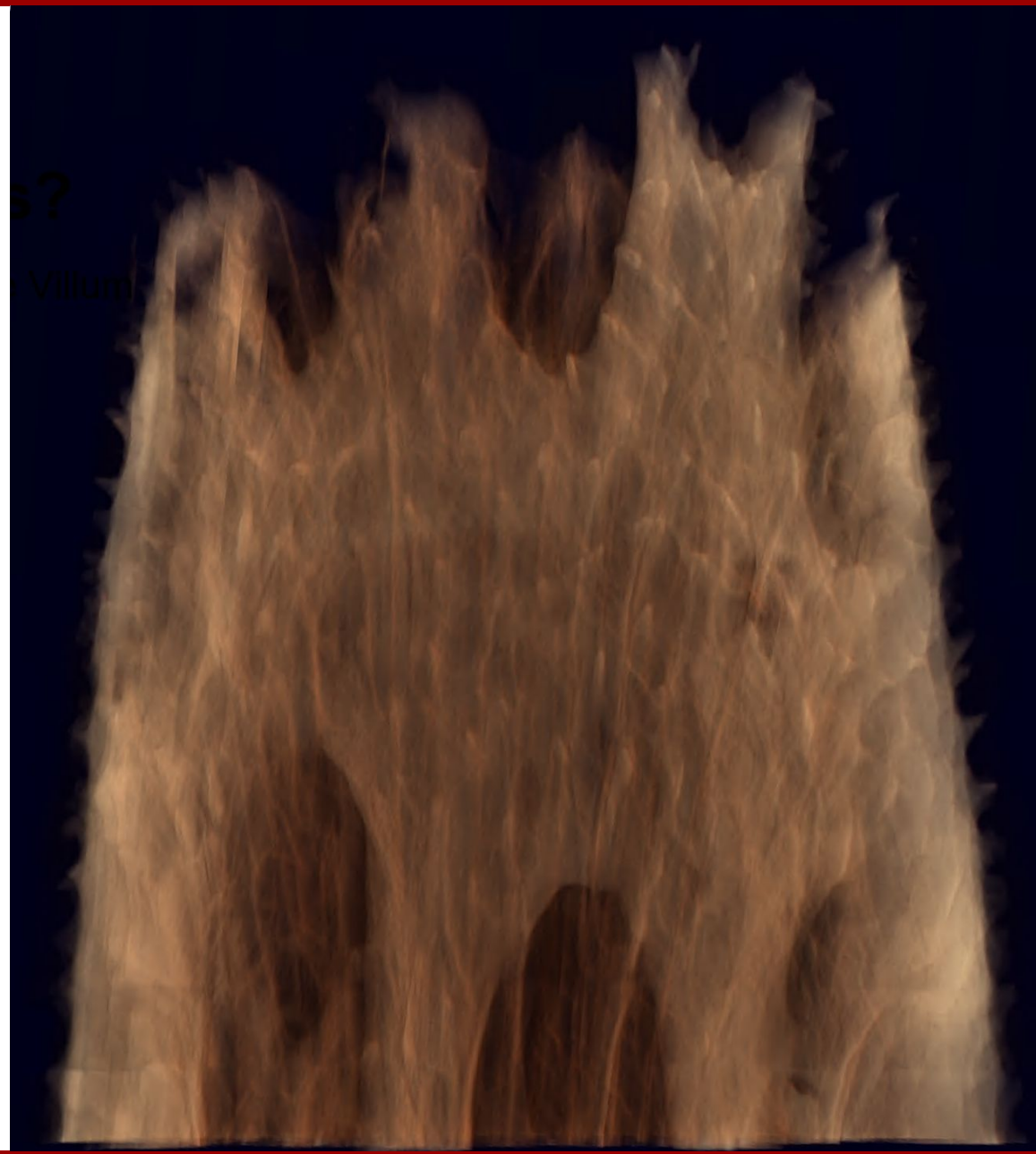
- Organic photovoltaic (OPV) cells have much potential as next-generation solar cells because they are lightweight, flexible and can be produced inexpensively.
- The external quantum efficiency (EQE) is significantly enhanced by absorption enhancement based on plasmonic scattering effects.
- Not only the scattering power but also the scattering direction must be considered, as the backward scattered power does not contribute to the absorption enhancement of the proposed OPVs.
- Optical simulation results provide solid evidence that the plasmonic *forward* light scattering effect greatly enhances the EQE and absorption of OPVs.



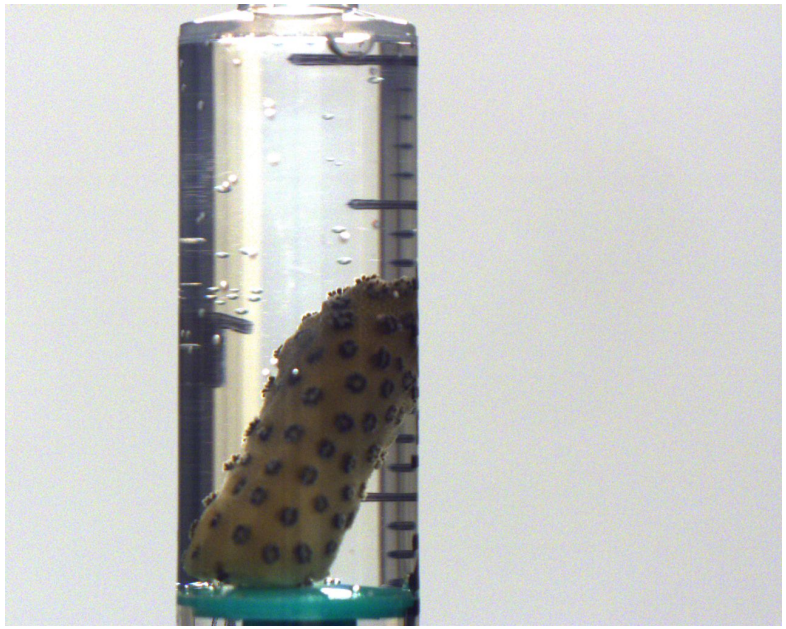
Slide from Frisvad and Khül about CORALS

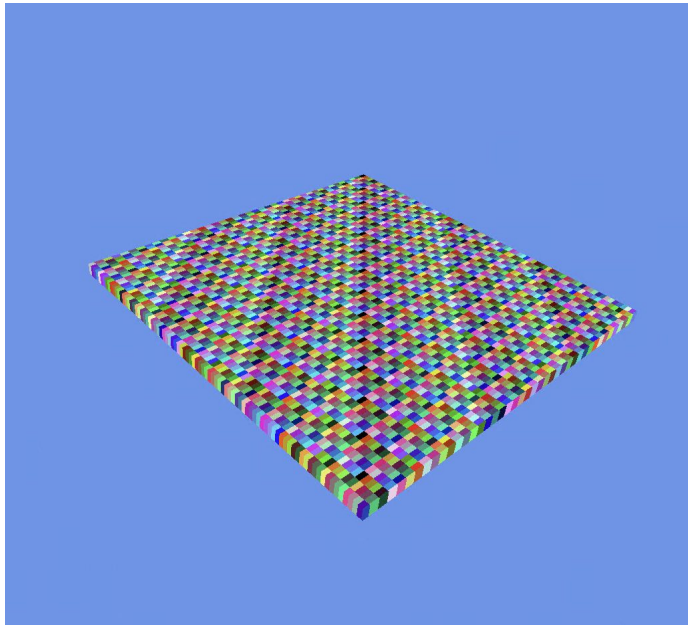
Why Corals?

- 30 GB large
- 1062x2664x2664
- 7,536,903,552 voxels
- 32 bit float in each voxel
- Around 28 GB



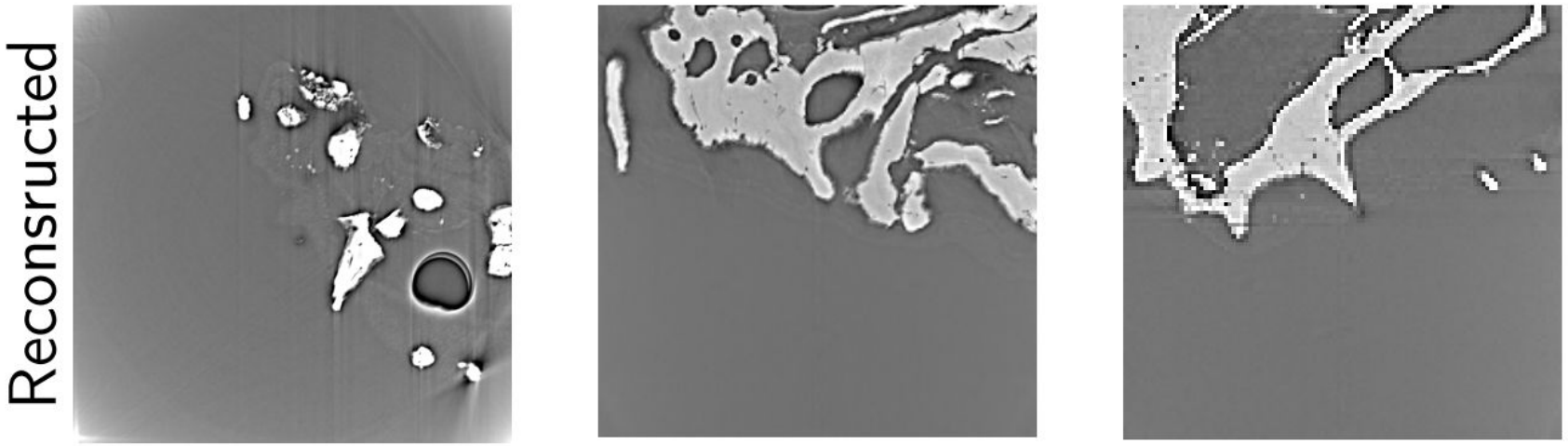
CORALS - New strategies for harvesting solar energy using coral-inspired microgeometry:



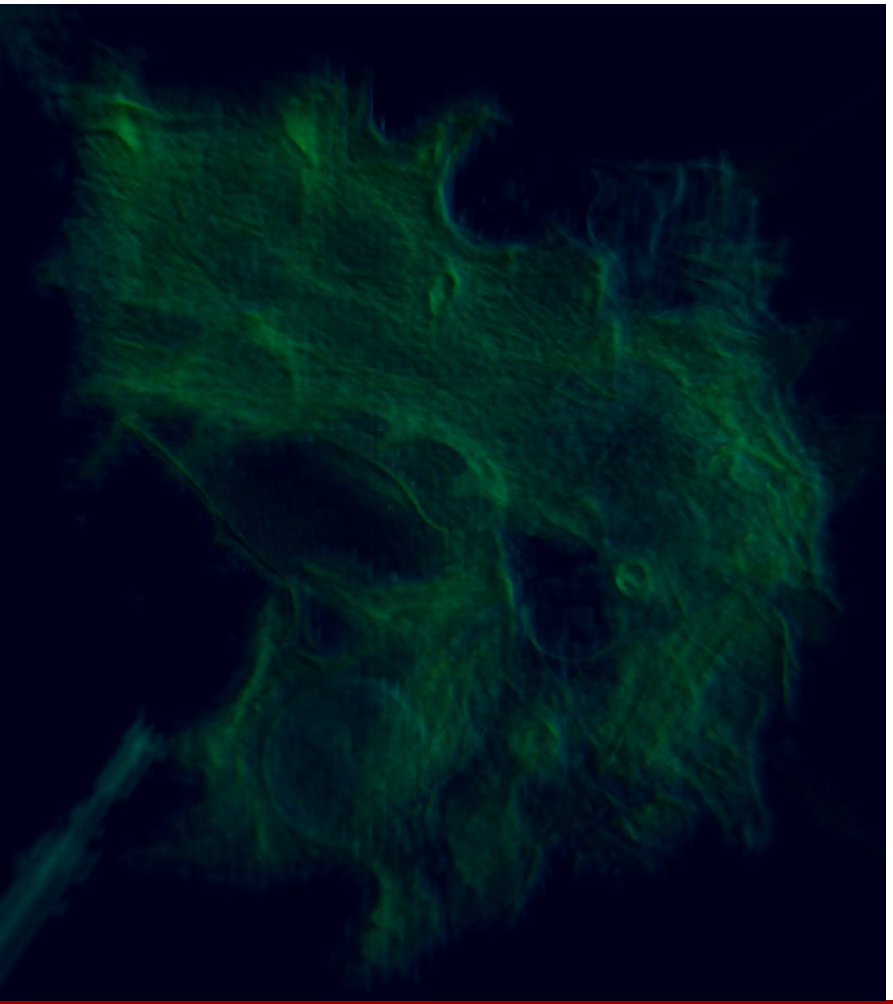


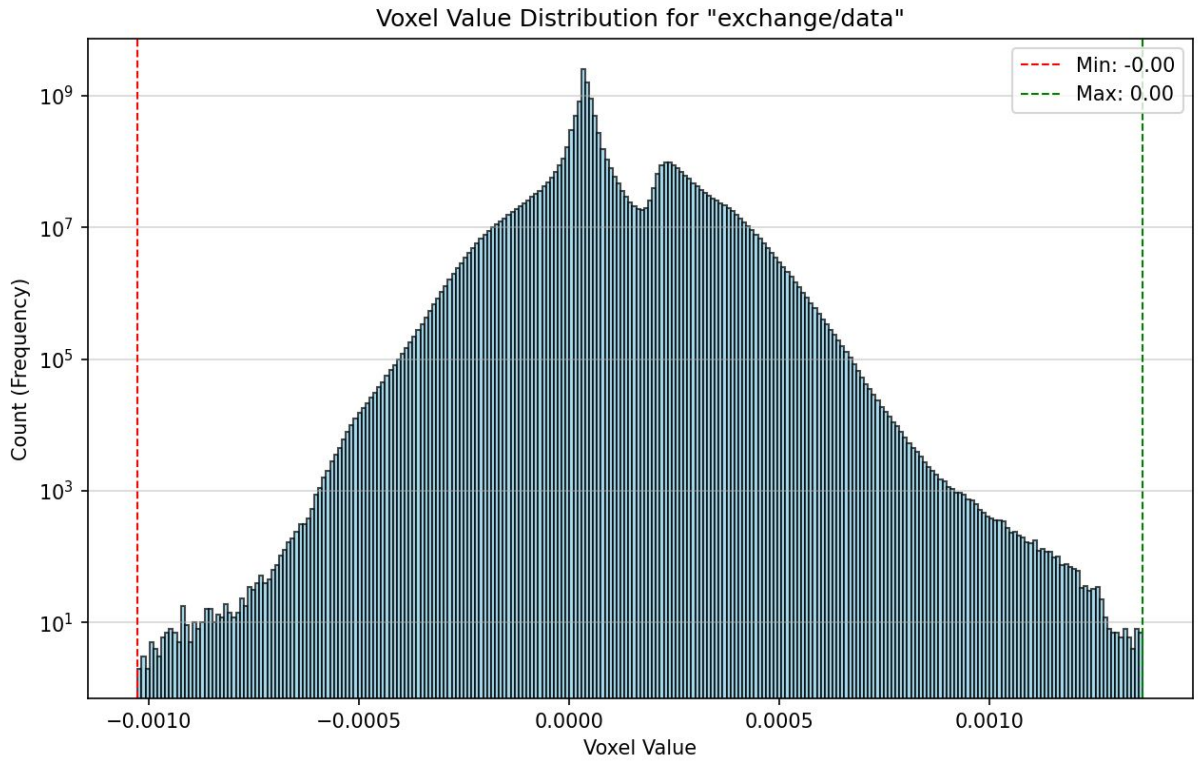
- File size: 40GB
- 2144x2176x2176
- 32 bit float in each voxel
- 10,151,788,544 voxels
- Around 37 GB

CORALS - New strategies for harvesting solar energy using coral-inspired microgeometry:

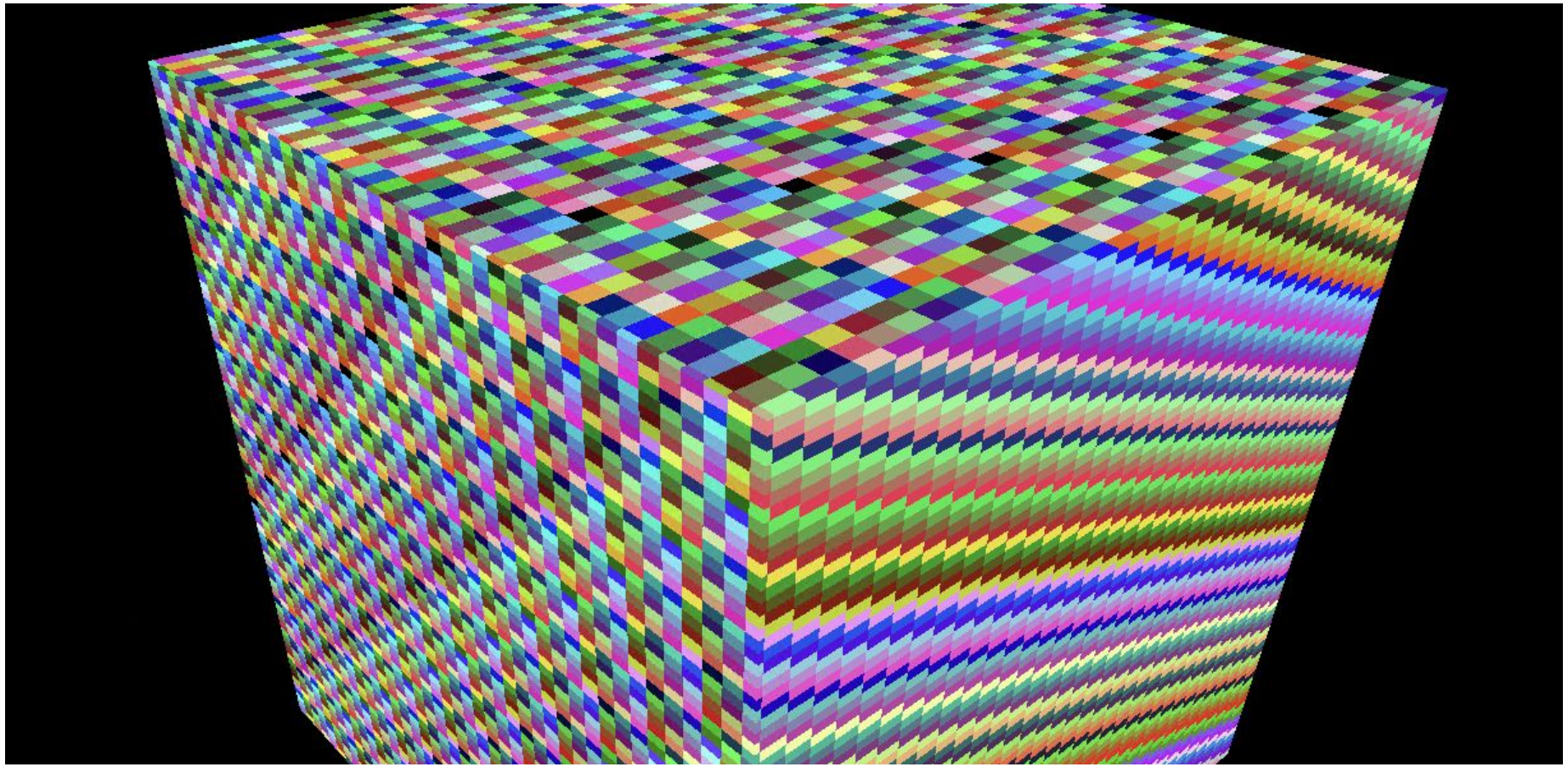


CORALS - New strategies for harvesting solar energy using coral-inspired microgeometry:



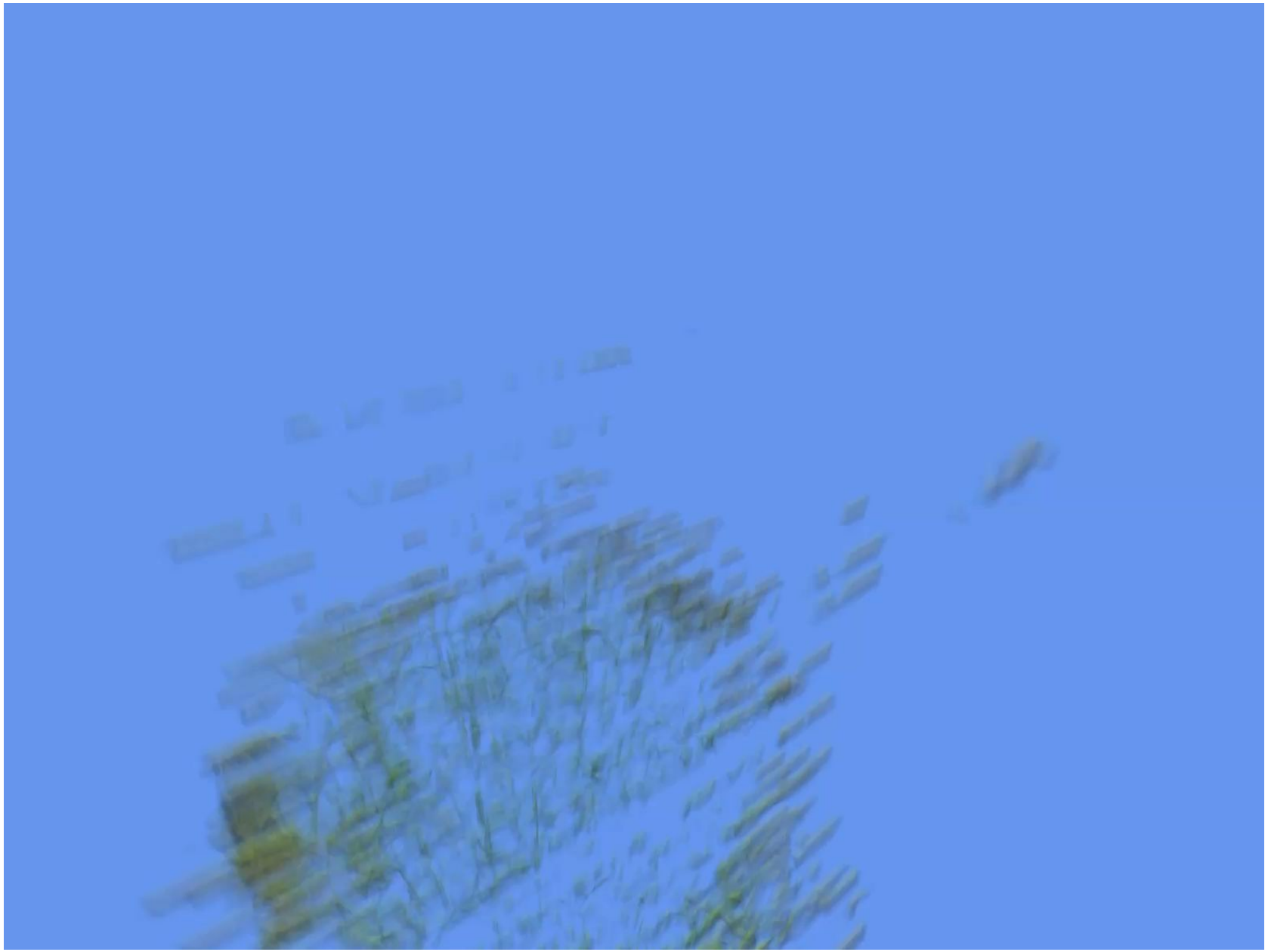


Volume data

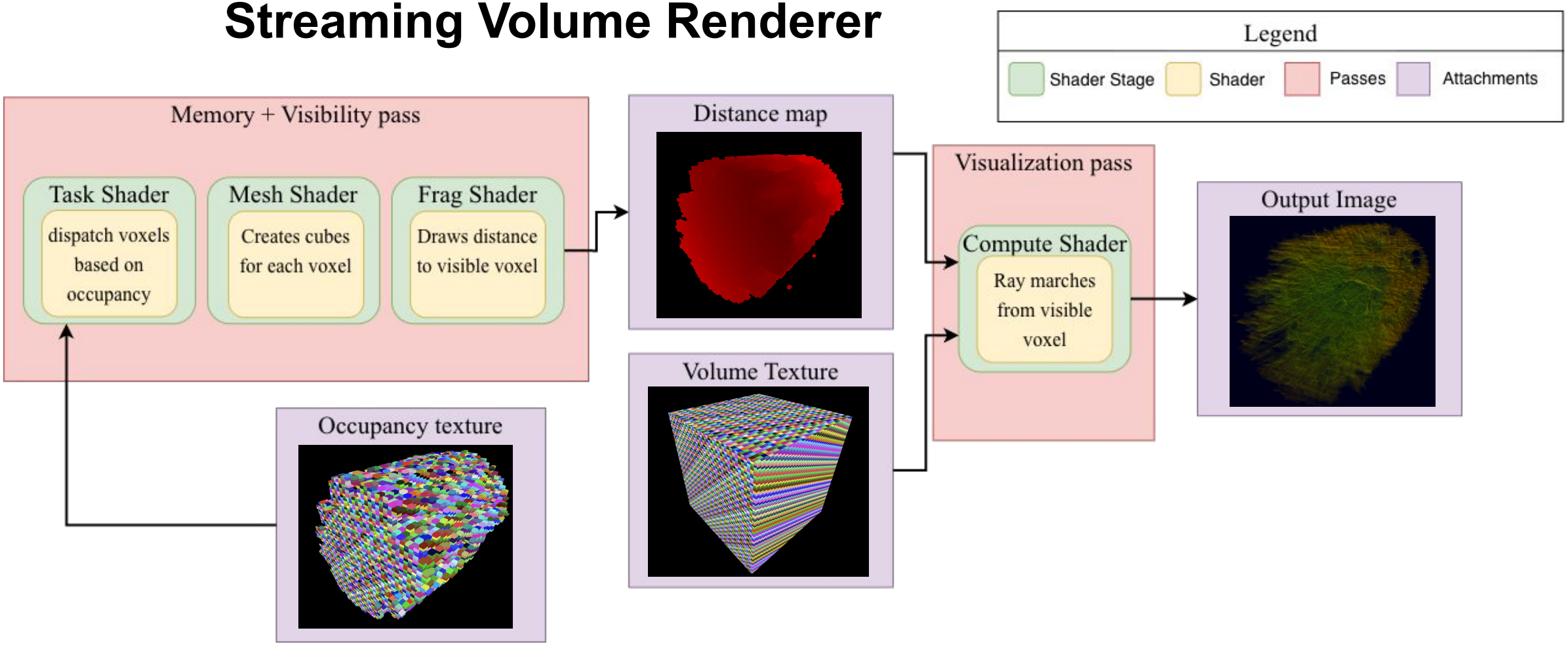



INDIANA JONES
LAST CRUSADE

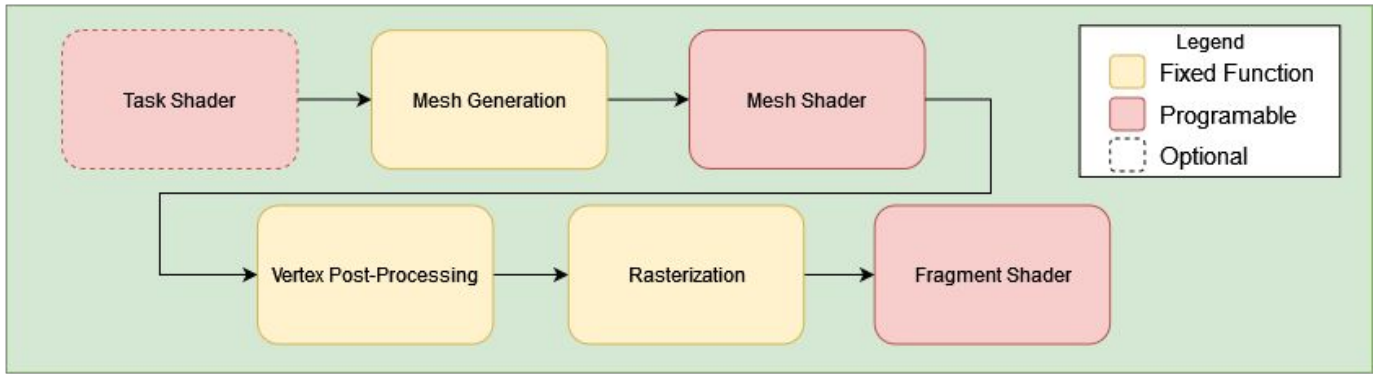

Windows
An error has occurred. To continue:
Press Enter to return to Windows, or
Press CTRL+ALT+DEL to restart your computer. If you do this,
you will lose any unsaved information in all open applications.
Error: 0E : 016F : BFF9B3D4
Press any key to continue _



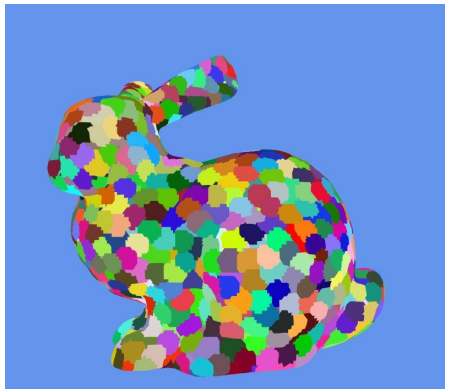
Streaming Volume Renderer



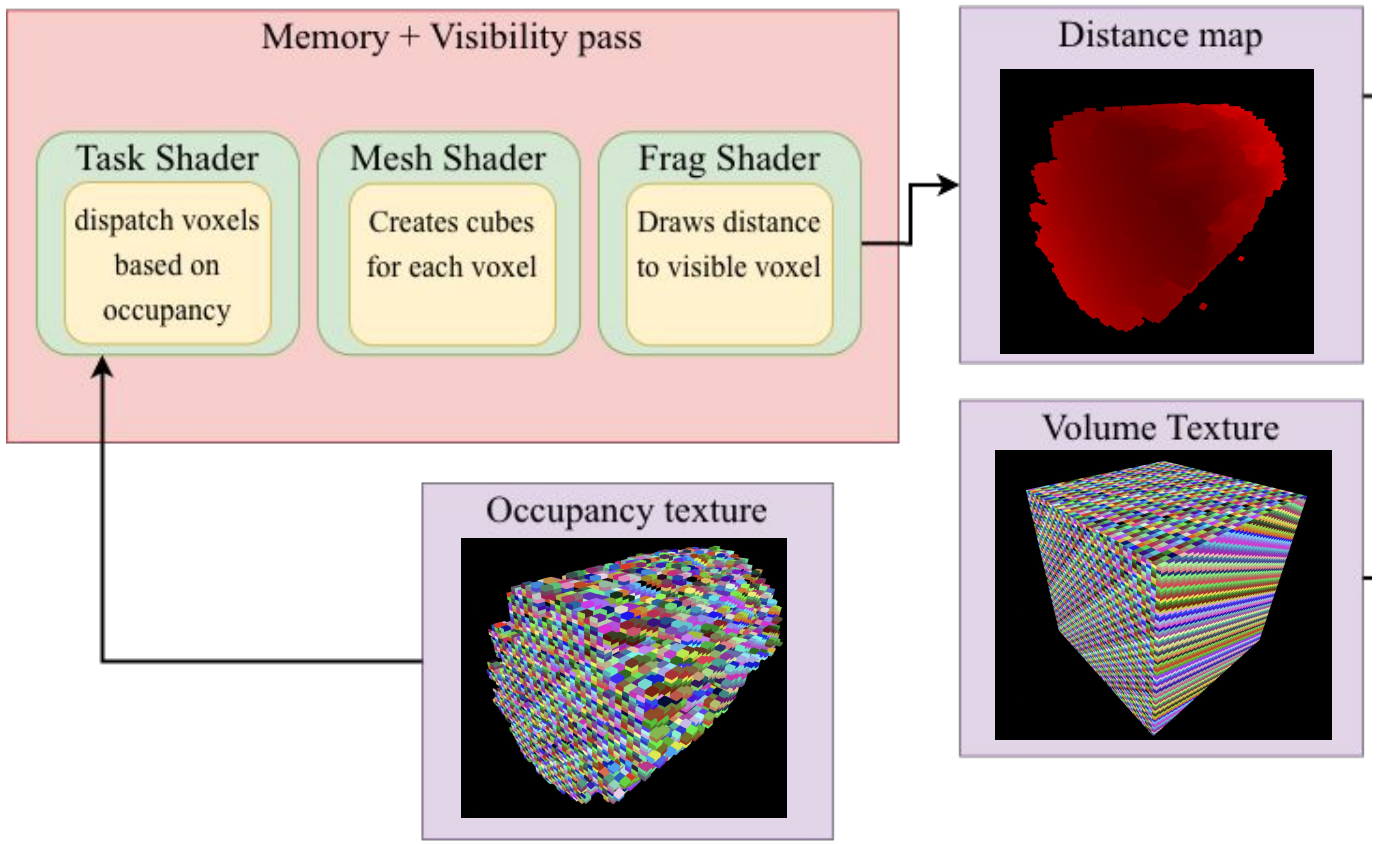
Streaming Volume Renderer



Mesh Shading Pipeline



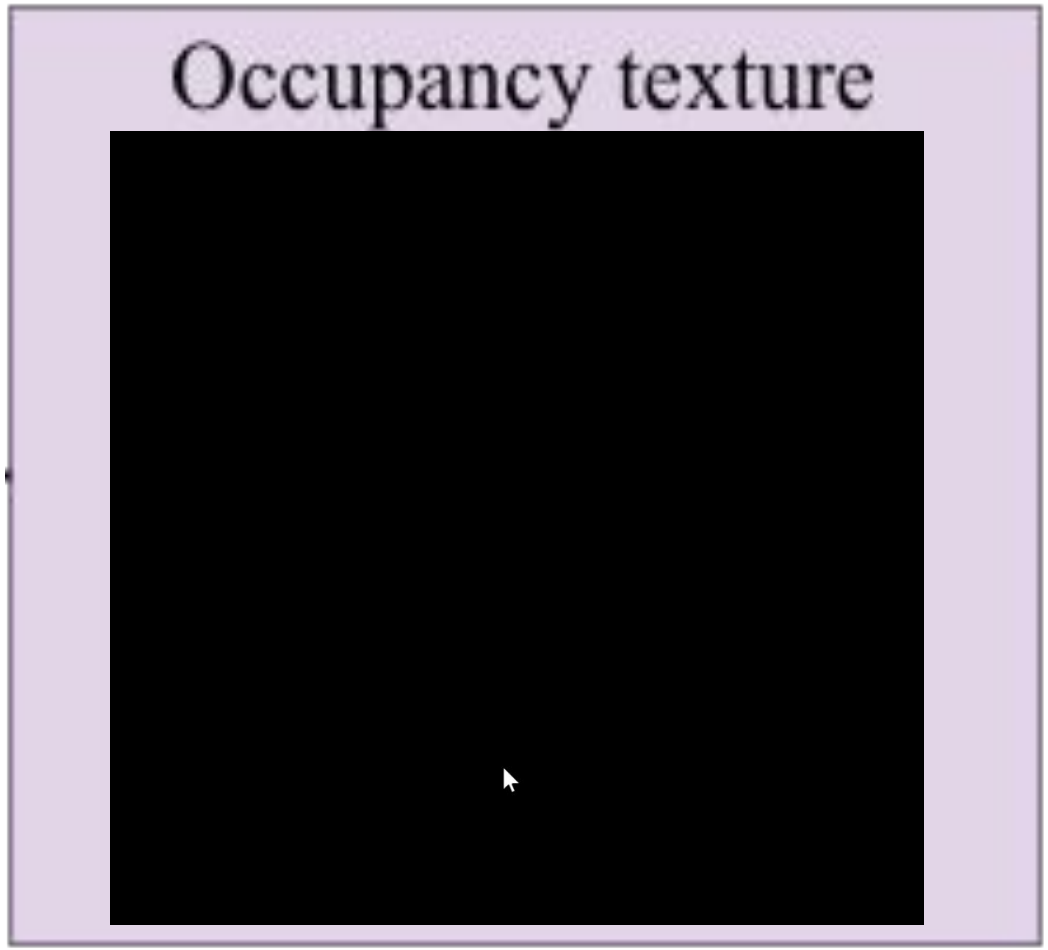
Streaming Volume Renderer



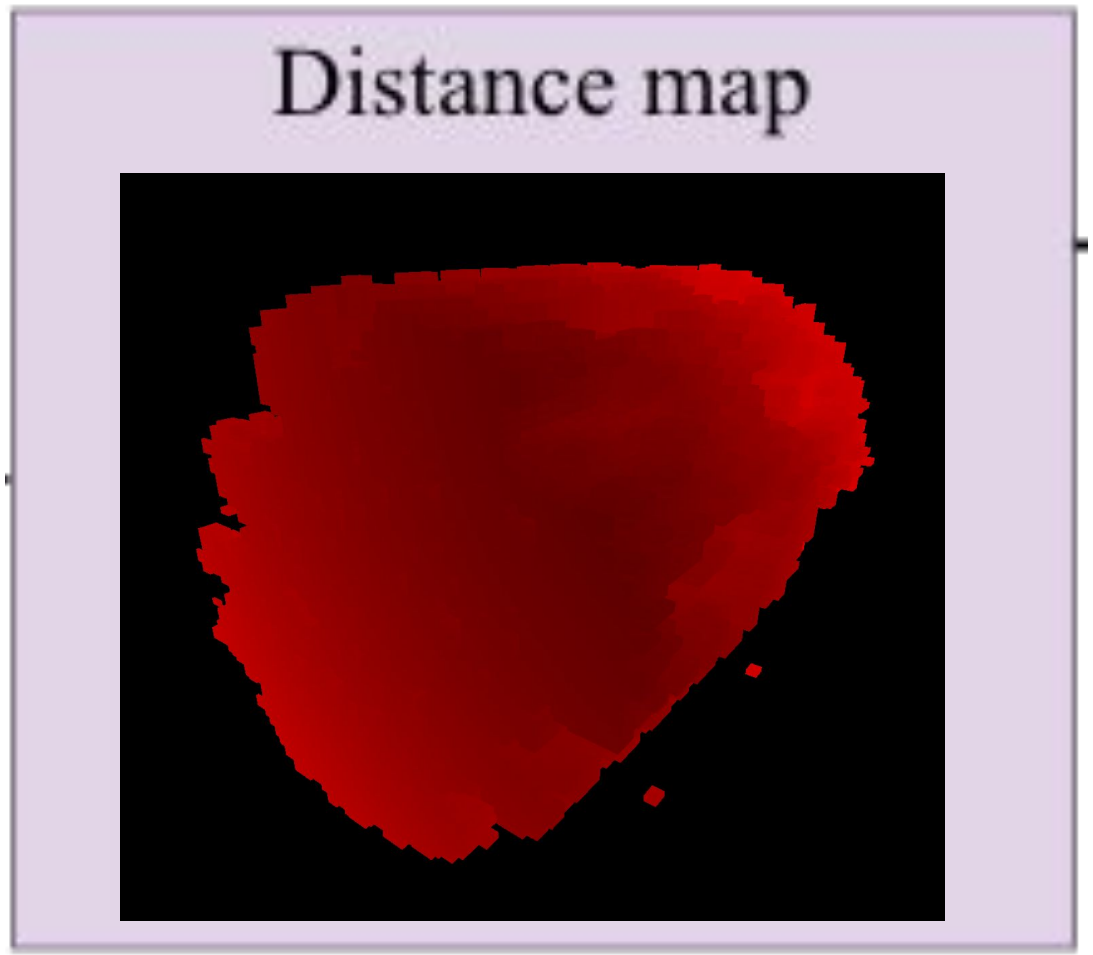
- **Pass 1: Entry Map Generation**
- **Task Shader**
  - Voxel loading/unloading
- **Mesh Shader**
  - Rasterizes the bounding boxes of active bricks.
- **Distance map**
  - Encodes the distance from the camera to the voxel in world space.
  - R32F

Task shader loading and unloading

- **Occupancy Texture**
  - 0 not visible
  - 1 unloaded
  - 2 visible (hot)
  - 3 visible (cold)
  - VK_FORMAT_R8
- **SSBO Buffer**
  - load + unload count
  - Lists of voxel Ids for each

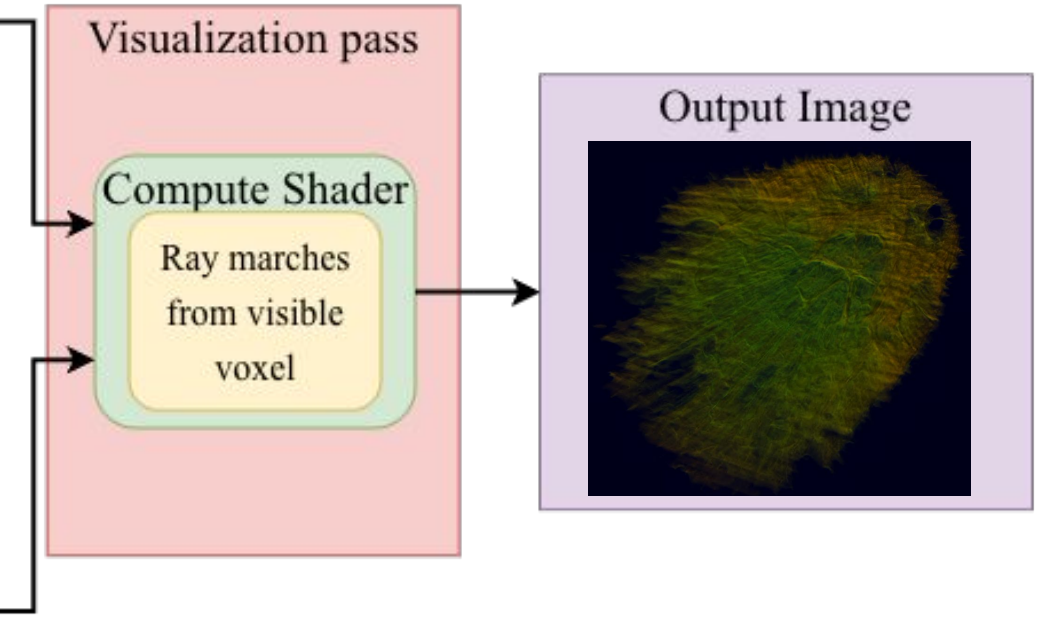


Mesh shader



- World distance to closest voxel
- Effectively allowing the compute pass to skip the empty space
- VK_FORMAT_R8

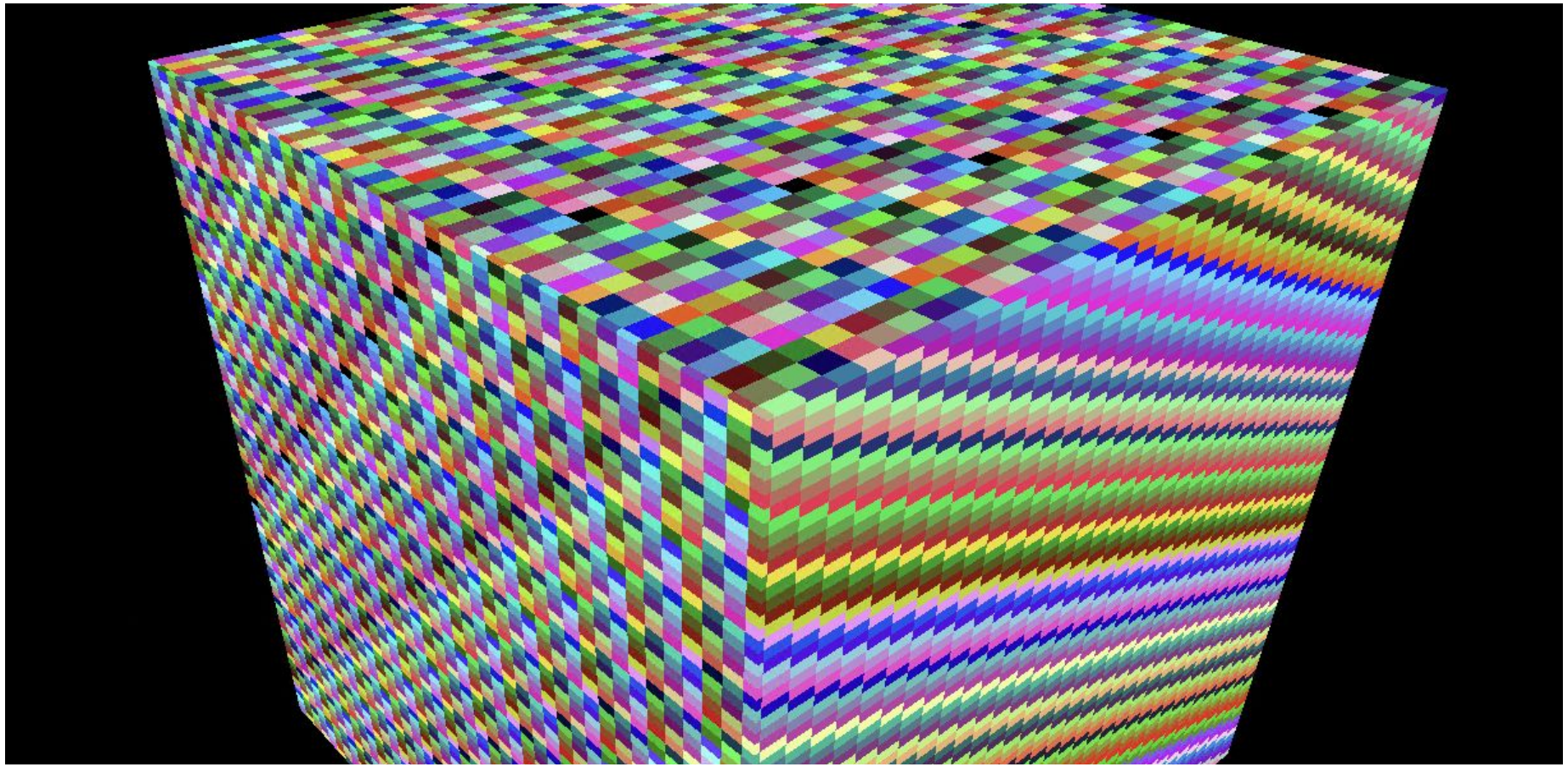
Compute-based Raymarching



- Uses distance map as starting point for ray marching
- Acummulates attenuation in alpha channel
- Ends when alpha reaches 1.0 or exits volume on the backside.

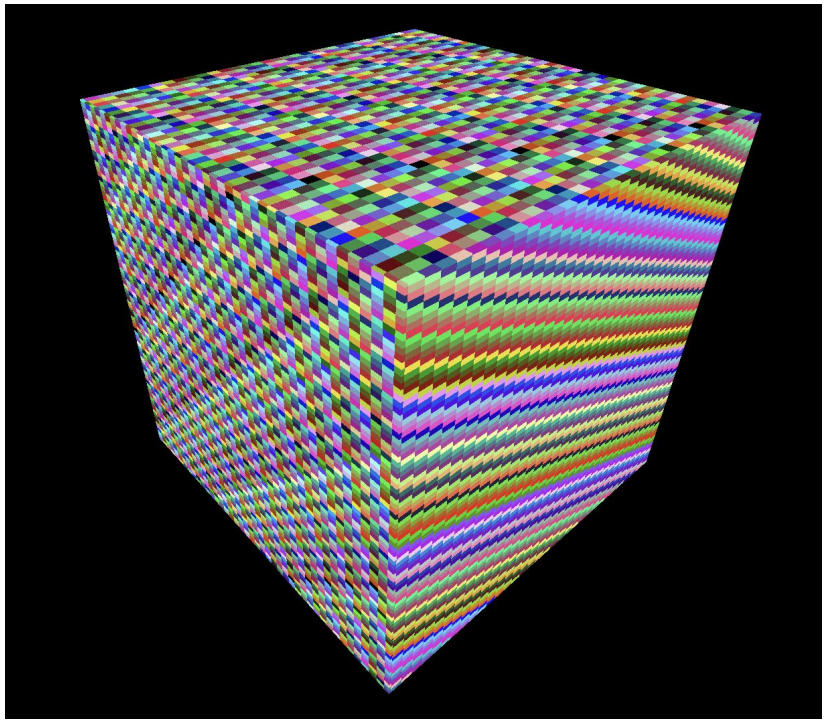
$$Coordinate_{normalized} = \left(\frac{Offset_{voxel}}{Total_{voxels}}\right) - 0.5$$

Volume data in Vulkan



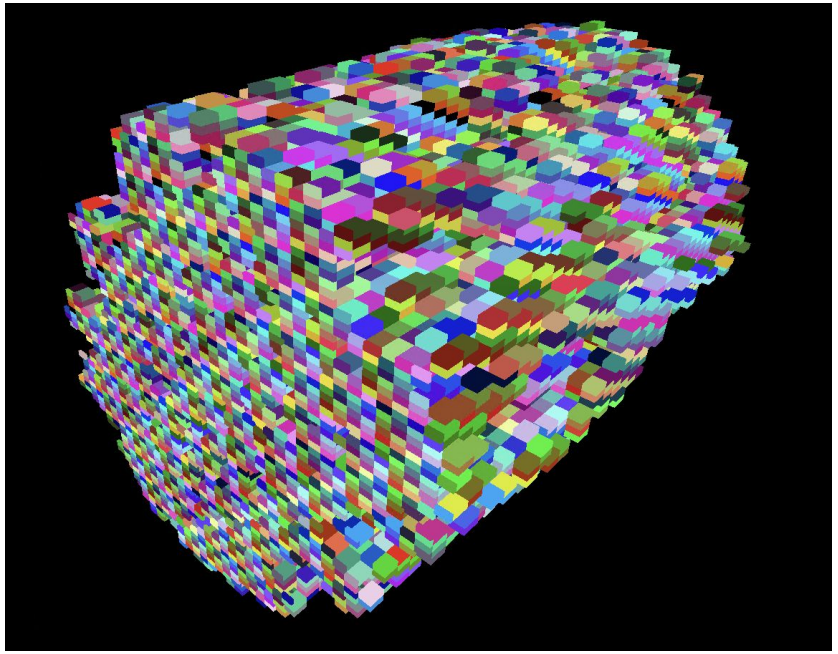
Volume data in Vulkan

```
VkImageCreateInfo imgInfo{};
imgInfo.sType = VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO;
imgInfo.imageType = VK_IMAGE_TYPE_3D;
imgInfo.extent = {mesh->textures["volumeTex"].width,
                  mesh->textures["volumeTex"].height,
                  mesh->textures["volumeTex"].depth};
imgInfo.mipLevels = mesh->textures["volumeTex"].mipLevels;
imgInfo.arrayLayers = 1;
imgInfo.format = mesh->textures["volumeTex"].format;
imgInfo.tiling = VK_IMAGE_TILING_OPTIMAL;
imgInfo.initialLayout = VK_IMAGE_LAYOUT_UNDEFINED;
imgInfo.usage = VK_IMAGE_USAGE_SAMPLED_BIT | VK_IMAGE_USAGE_TRANSFER_DST_BIT;
imgInfo.sharingMode = VK_SHARING_MODE_EXCLUSIVE;
imgInfo.samples = VK_SAMPLE_COUNT_1_BIT;
```



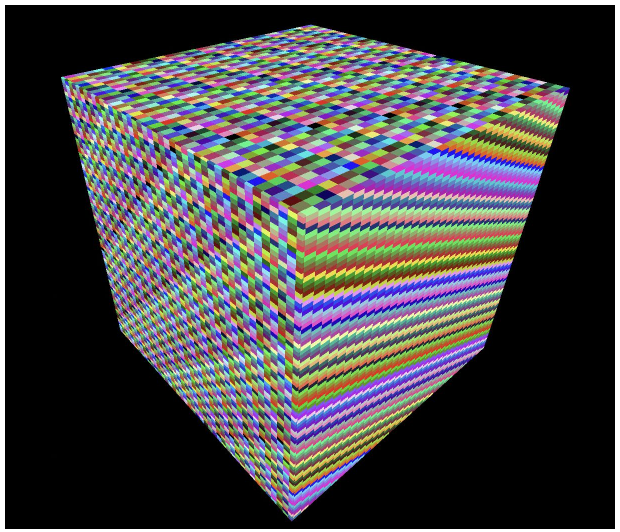
Volume data in Vulkan

```
VkImageCreateInfo imgInfo{};
imgInfo.sType = VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO;
imgInfo.imageType = VK_IMAGE_TYPE_3D;
imgInfo.extent = {mesh->textures["volumeTex"].width,
                  mesh->textures["volumeTex"].height,
                  mesh->textures["volumeTex"].depth};
imgInfo.mipLevels = mesh->textures["volumeTex"].mipLevels;
imgInfo.arrayLayers = 1;
imgInfo.format = mesh->textures["volumeTex"].format;
imgInfo.tiling = VK_IMAGE_TILING_OPTIMAL;
imgInfo.initialLayout = VK_IMAGE_LAYOUT_UNDEFINED;
imgInfo.usage = VK_IMAGE_USAGE_SAMPLED_BIT | VK_IMAGE_USAGE_TRANSFER_DST_BIT;
imgInfo.sharingMode = VK_SHARING_MODE_EXCLUSIVE;
imgInfo.samples = VK_SAMPLE_COUNT_1_BIT;
imgInfo.flags =
    VK_IMAGE_CREATE_SPARSE_BINDING_BIT | VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT;
```

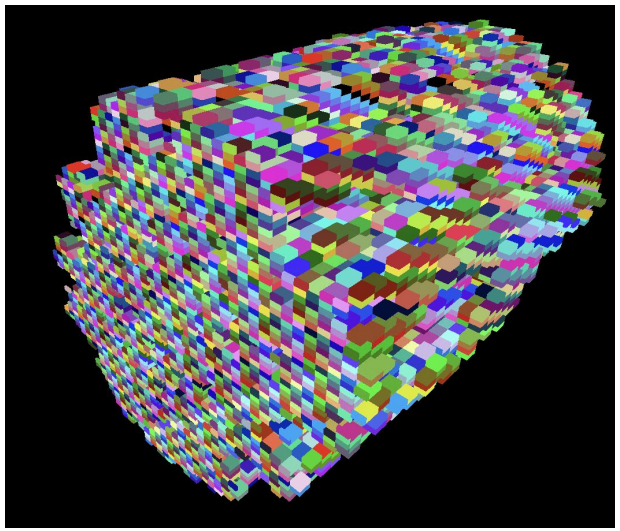


- `VK_IMAGE_CREATE_SPARSE_BINDING_BIT`
  - Allows the image to be backed by non-contiguous memory regions.
- `VK_IMAGE_CREATE_SPARSE_RESIDENCY_BIT`
  - Allows the image to be partially backed

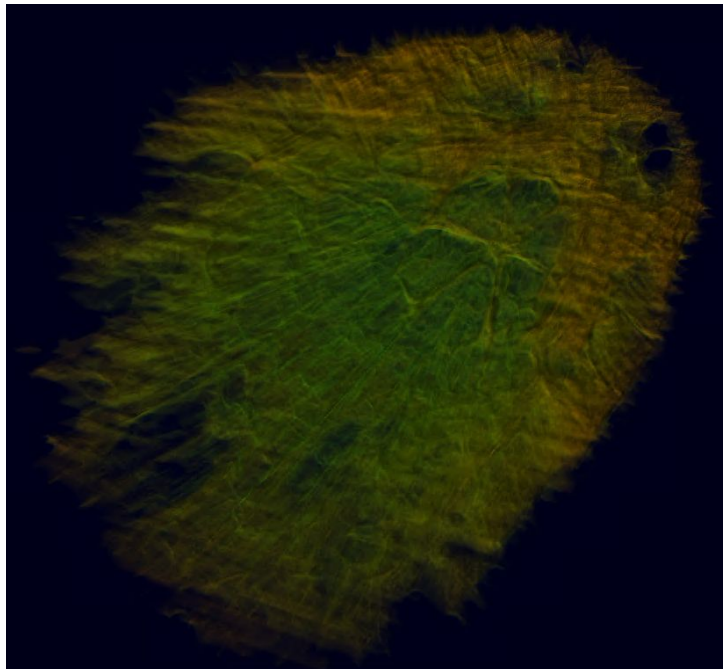
Volume data



119952 / 119952 Bricks



58442 / 119952 Bricks



Same coral from before with 7,536,903,552 voxels

Virtual Bricking

The tileSize defines the smallest possible “brick” size that be used for the virtual bricks.
We create one 3D texture for where each texel points to one virtual Brick.
On an NVIDIA RTX 4090 the pagesize is: 64x32x32

```
uint32_t count = 0;
vkGetImageSparseMemoryRequirements(
pDevice->Device(), mesh->textures["volumeTex"].image, &count, nullptr);


std::vector<VkSparseImageMemoryRequirements> reqs(count);
vkGetImageSparseMemoryRequirements(pDevice->Device(),
mesh->textures["volumeTex"].image, &count,
reqs.data());

VkExtent3D tileSize = reqs[0].formatProperties.imageGranularity; pageSizeBytes =
tileSize.width * tileSize.height * tileSize.depth * bytesPerVoxel;
```

Virtual Bricking

The size of the virtual bricks can be used for memory allocation, and because of `VK_IMAGE_CREATE_SPARSE_BINDING_BIT` these do not have to be contiguous

```
VkExtent3D tileSize = reqs[0].formatProperties.imageGranularity; pageSizeBytes =
tileSize.width * tileSize.height * tileSize.depth * bytesPerVoxel;
```

Divide memory chunks into pages. With each page being the size of a virtual brick

```
// Slice chunk into pages...
VkDeviceSize currentOffset = 0;
while (currentOffset + pageSizeBytes <= CHUNK_SIZE) {
    freePages.push_back({block, currentOffset});
    VkDeviceSize nextStart = currentOffset + pageSizeBytes;
    currentOffset = (nextStart + alignment - 1) / alignment * alignment;
}
```

CPU-side pre-processing

- 32 bit values but range is between -0.0010253082728013396 and 0.0013644369319081306Qantization to 16 bits would make each step cover 0.0000000365.

$$Integer = \text{clamp}\left(\text{round}\left((x - x_{min}) \times \frac{2^N - 1}{x_{max} - x_{min}}\right)\right)$$

- Remove empty bricks

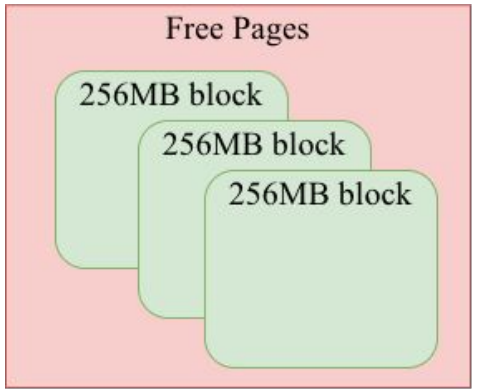
```
for (int z = 0; z < bricksZ; ++z) {
  for (int y = 0; y < bricksY; ++y) {
    for (int x = 0; x < bricksX; ++x) {
        int brickIndex = x + (y * bricksX) + (z * bricksX * bricksY);
       if (isBrickEmpty(brickIndex) continue;
        occupancyData[brickIndex] = 1;
    }
  }
}
```

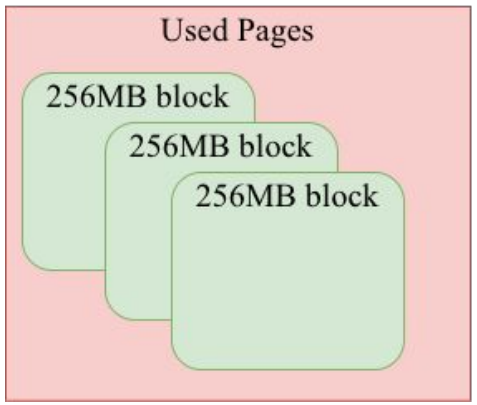
Is Brick Empty Test

```
template <typename T>
bool isBrickEmptyTemplate(const T *data, int bx, int by, int bz, int bWidth, int bHeight, int bDepth, int totalWidth, int totalHeight,
T threshold) {
for (int z = 0; z < bDepth; z++) {
  // Calculate offset to the start of the current Z slice in the total volume
  size_t zOffset = static_cast<size_t>(bz + z) * totalWidth * totalHeight;

  for (int y = 0; y < bHeight; y++) {
    // Calculate offset to the start of the current row within the Z slice
    size_t yOffset = static_cast<size_t>(by + y) * totalWidth;

    // Calculate the exact starting index for this row
    size_t rowStart = zOffset + yOffset + bx;

    // Scan the row
      for (int x = 0; x < bWidth; x++) {
        if (data[rowStart + x] > threshold) {
          return false; // Found valid data! Keep this brick.
  }}}}
  return true; // Brick is completely empty
}
```

Per Frame Uploads

Per Frame Uploads





- **SSBO buffer:**
  - Upload and load count
  - Brick IDS
- **Upload Flow:**
  - **Eviction:**
    - Reclaiming bricks for unloading
  - **Upload:**
    - Upload new bricks and voxel data
  - **Update:**
    - Occupancy Texture

Per frame uploads - submitting

```
for (int z = 0; z < tileSize.depth; z++) {// --- COPY LOOP ---
  int globalZ = oz + z;
  if (globalZ >= fullDepth) continue; // Check Volume Depth

  for (int y = 0; y < tileSize.height; y++) {
    int globalY = oy + y;
    if (globalY >= fullHeight) continue; // Check Volume Height

    int globalX = ox;
    int remainingX = fullWidth - globalX;
    int remainingBytes = remainingX * bytesPerVoxel;
    int bytesToCopy = std::min(rowSizeBytes, remainingBytes);

    size_t srcPixelIndex = ((size_t)globalZ * fullWidth * fullHeight) +
    ((size_t)globalY * fullWidth) + globalX;
    size_t dstPixelIndex = (z * streamingCtx.tileSize.height * streamingCtx.tileSize.width) +
(y * streamingCtx.tileSize.width);

    std::memcpy(dstVoxelPtr + (dstPixelIndex * bytesPerVoxel), srcVolume + (srcPixelIndex *  bytesPerVoxel), bytesToCopy);
}}
```

Per frame uploads - submitting

```
// VOXEL COPY
VkBufferImageCopy voxelCopy = {};
voxelCopy.bufferOffset = currentOffset;
voxelCopy.imageSubresource = {VK_IMAGE_ASPECT_COLOR_BIT, 0, 0, 1};
voxelCopy.imageOffset = {(int32_t)ox, (int32_t)oy, (int32_t)oz};


// CALCULATE UPLOAD SIZE
uint32_t safeW = std::min((uint32_t)streamingCtx.tileSize.width, (uint32_t)(fullWidth - ox));
uint32_t safeH = std::min((uint32_t)streamingCtx.tileSize.height, (uint32_t)(fullHeight - oy));
uint32_t safeD = std::min((uint32_t)streamingCtx.tileSize.depth, (uint32_t)(fullDepth - oz));
voxelCopy.imageExtent = {safeW, safeH, safeD};
voxelCopies.push_back(voxelCopy);
```

Per frame uploads - submitting

```
// UPDATE STATE
uint32_t *statePtr = (uint32_t *)(dstVoxelPtr + voxelSizeBytes);
*statePtr = RESIDENT_HOT;

VkBufferImageCopy stateCopy = {};
stateCopy.bufferOffset = currentOffset + voxelSizeBytes;
stateCopy.imageSubresource = {VK_IMAGE_ASPECT_COLOR_BIT, 0, 0, 1};
stateCopy.imageOffset = {(int32_t)bX, (int32_t)bY, (int32_t)bZ};
stateCopy.imageExtent = {1, 1, 1};
stateCopies.push_back(stateCopy);
```

Per frame uploads - submitting

```
// BIND MEMORY
VkSparseImageMemoryBindInfo imgBindInfo = {};
imgBindInfo.image = mesh->textures["volumeTex"].image;
imgBindInfo.bindCount = (uint32_t)binds.size();
imgBindInfo.pBinds = binds.data();


VkBindSparseInfo bindInfo = {VK_STRUCTURE_TYPE_BIND_SPARSE_INFO};
bindInfo.imageBindCount = 1;
bindInfo.pImageBinds = &imgBindInfo;


vkQueueBindSparse(m_pDevice->m_pGraphicsQueue, 1, &bindInfo,
VK_NULL_HANDLE);
```

Memory model without streaming

```
for (int z = 0; z < bricksZ; ++z) {
  for (int y = 0; y < bricksY; ++y) {
    for (int x = 0; x < bricksX; ++x) {

      if (isBrickEmpty(data, ox, oy, oz, cWidth, cHeight,
      cDepth, width, height, fmt, treshold)) {
        continue; // Skip allocation and binding
      }

      int brickIndex = x + (y * bricksX) + (z * bricksX * bricksY);
      occupancyData[brickIndex] = 1;

      // ADD MEMORY BINDING OR AABB INFO FOR TLAS/BLAS
    }
  }
}
```

Same memory model can be used for ray tracing and generally allows for quite large volumes without streaming.

Memory model for Ray Tracing

```
// CREATE AABB POSITION FOR BLAS
float nMinX = ((float)ox * invWidth) - 0.5f;
float nMinY = ((float)oy * invHeight) - 0.5f;
float nMinZ = ((float)oz * invDepth) - 0.5f;

float nMaxX = ((float)(ox + currentWidth) * invWidth) - 0.5f;
float nMaxY = ((float)(oy + currentHeight) * invHeight) - 0.5f;
float nMaxZ = ((float)(oz + currentDepth) * invDepth) - 0.5f;

VkAabbPositionsKHR aabb{nMinX, nMinY, nMinZ, nMaxX, nMaxY, nMaxZ};
mesh->aabbs.push_back(aabb);
// PER AABB INFO FOR SSBO
aabbInfo info;
info.offset = glm::vec4(ox, oy, oz, 0.0);

info.extent = glm::vec4(currentWidth, // Width
currentHeight, // Height
currentDepth, // Depth
0 // Atlas Index);

mesh->aabbInfos.push_back(info);
```

Future work

- NanoVDB for more swiss cheese volumes and better Brick traversal
- Using the distance map as visibility map for 2xpass brick culling
- UI for biologists to control x-ray transfer function
- Your input

THANK YOU

Code will be on github.com/senbyo within a month