Real-Time High-Resolution Sparse Voxelization with Application to Image-Based Modeling

Charles Loop Cha Zhang Zhengyou Zhang

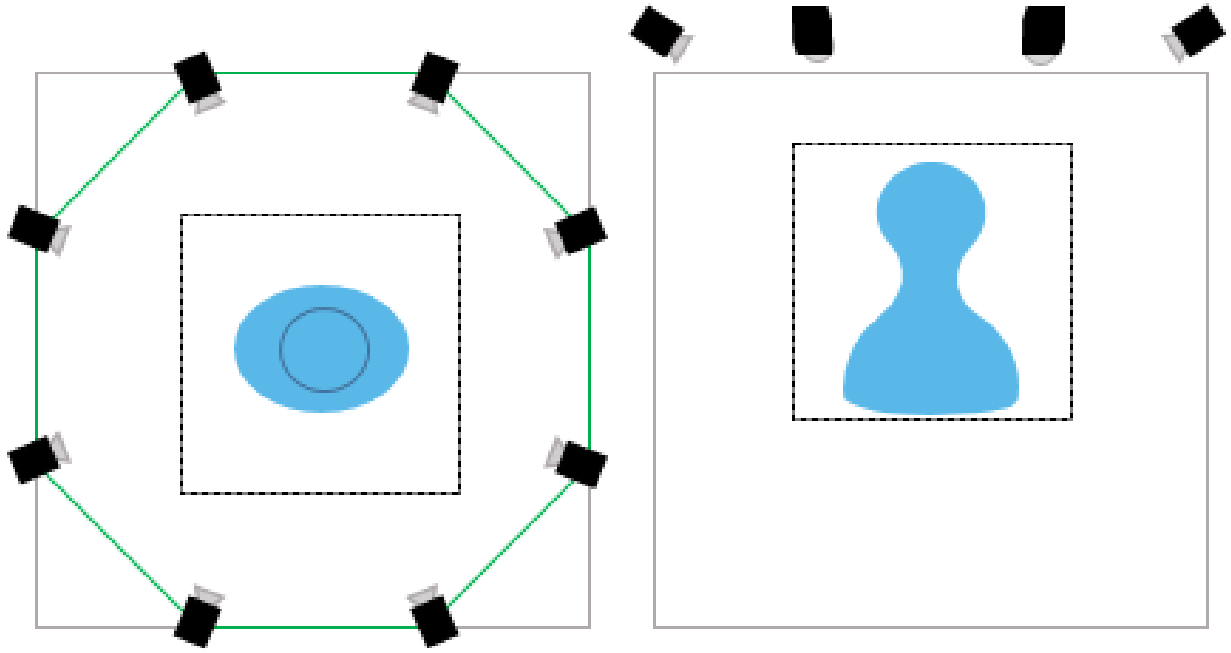
Microsoft Research

Goal: live free-viewpoint visualization of remote scenes

- Scene capture from calibrated multi-sensor input
- Current depth sensors technology issues:
  - Multi-sensor interference
  - Lack synchronization
  - IR light scattering on hair
- Use RGB cameras plus green-screen instead
  - Smooth visual hull reconstruction from silhouettes
  - Inaccurate geometry
- Validate voxel-based approach
  - Massively parallel GPUs enables sparse, high resolution approach

Capture Rig

- 8 Point Grey Flea 2 Cameras
- Portrait Orientation
- 800x600 resolution
- 30 Hz



top view

side view

Outline

- GPU Based Sparse Voxelization
  - Data structure + Processing steps
- Smooth Visual Hull Reconstruction
  - Product of Silhouette Images
- Results and Conclusions

GPU-Based Sparse Voxelization

Voxel Representation

- Data Structure:

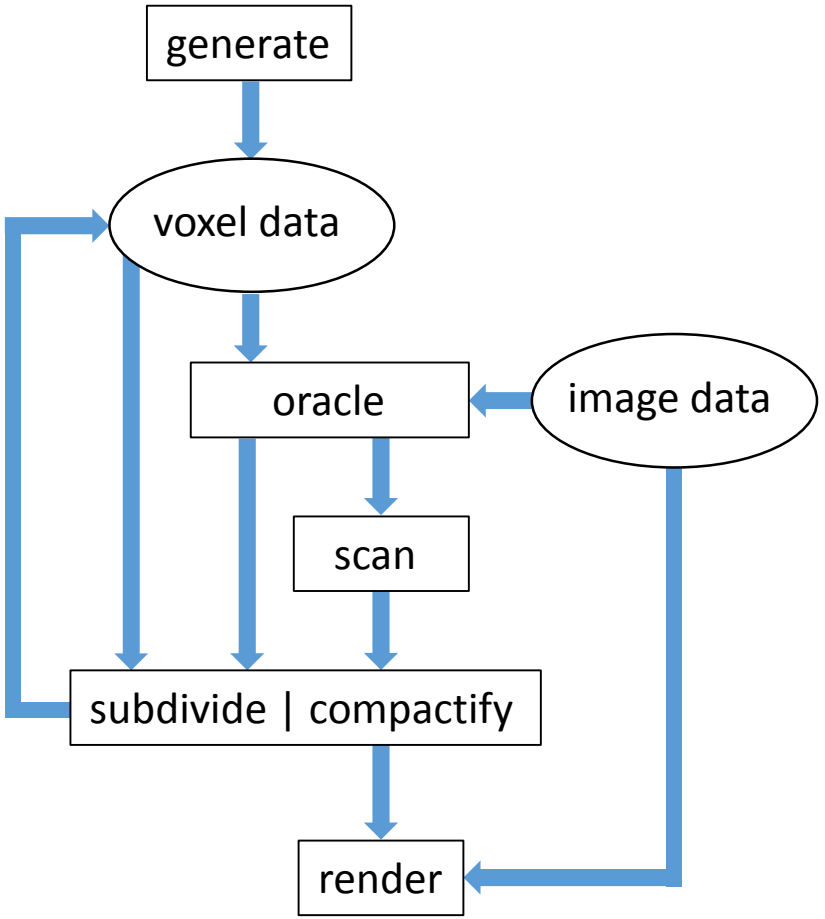
```
struct voxel
{
    unsigned int ix, iy, iz;
}
```

```
voxel VoxelList[MAX_VOXELS];
```

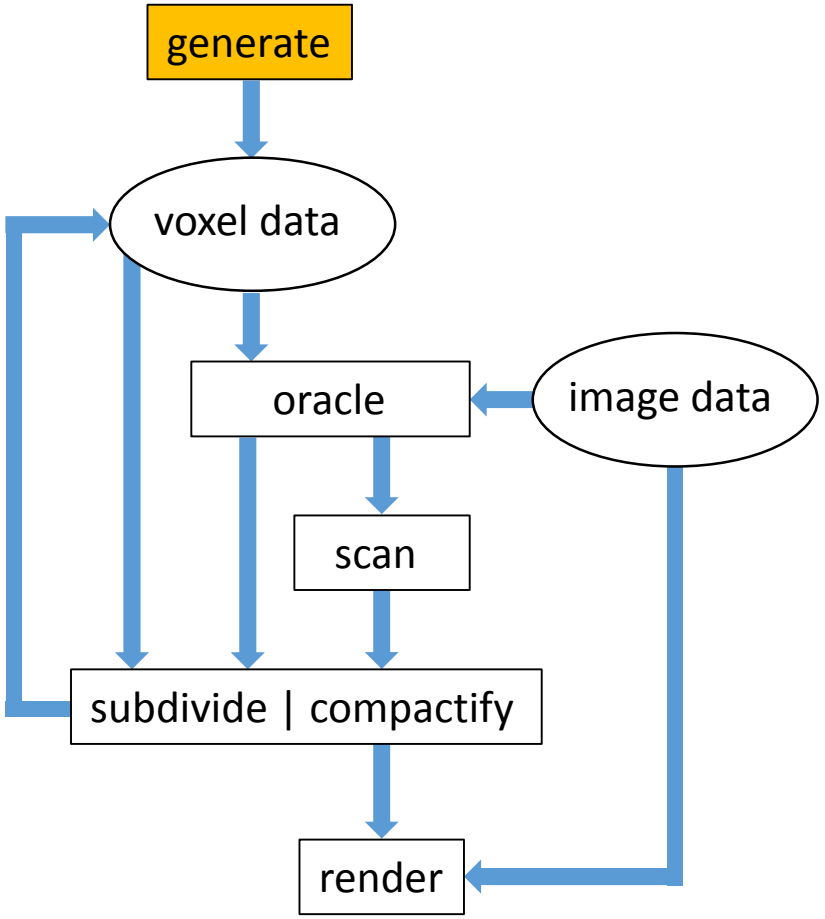
$\mathrm{ix}, \mathrm{iy}, \mathrm{iz} \in \{0, \ldots, 2^k\}$

```
unsigned int k; // hierarchy level
```

Processing Steps



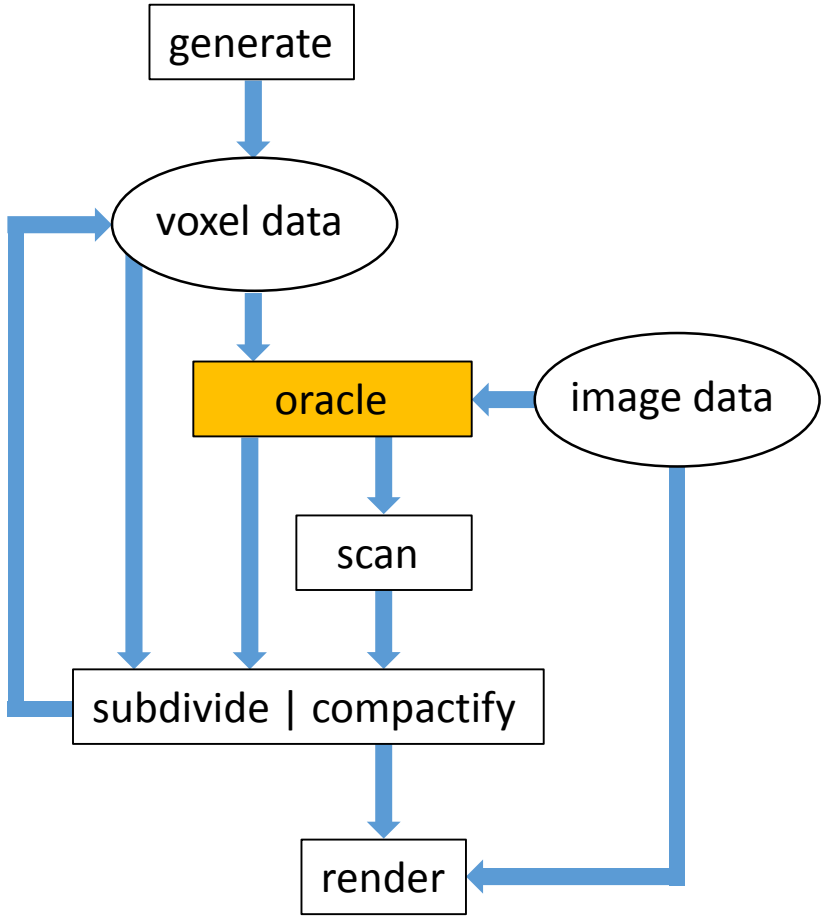
Generate



Begin at level `k=4`

```
voxel v;
v.ix = threadID/256;
v.iy = (threadID/16)%16;
v.iz = threadID%16;
VoxelList[threadID] = v;
```

Oracle




```
oracleOutput[threadID] = DECISION;
```

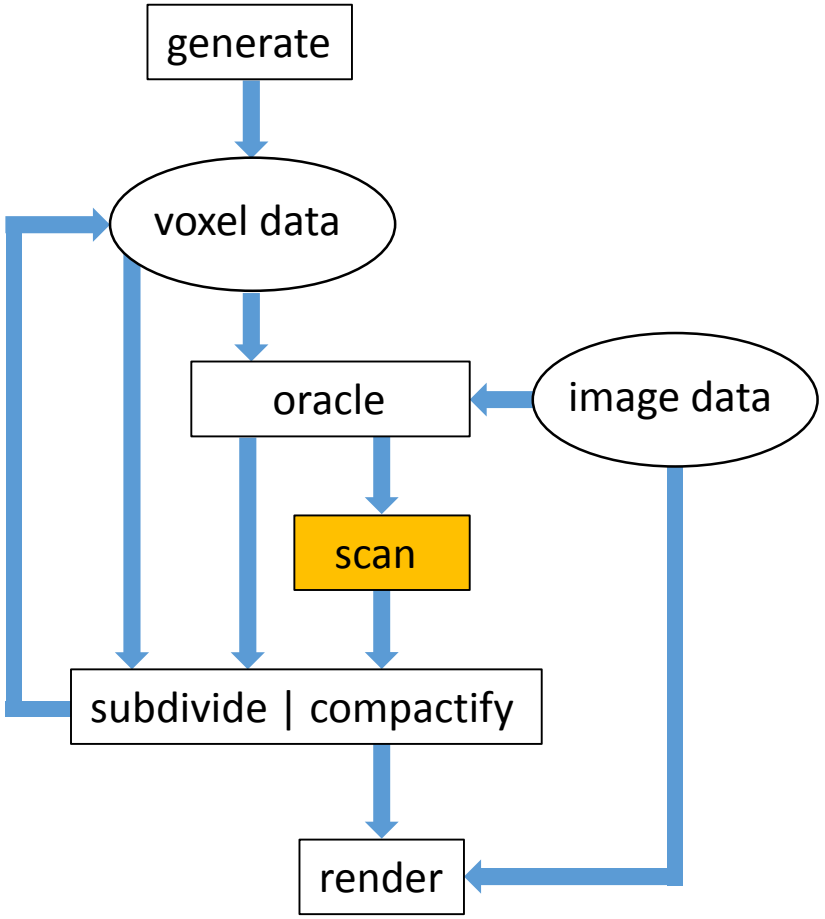
Binary decision:

```
#define CULL 0
```

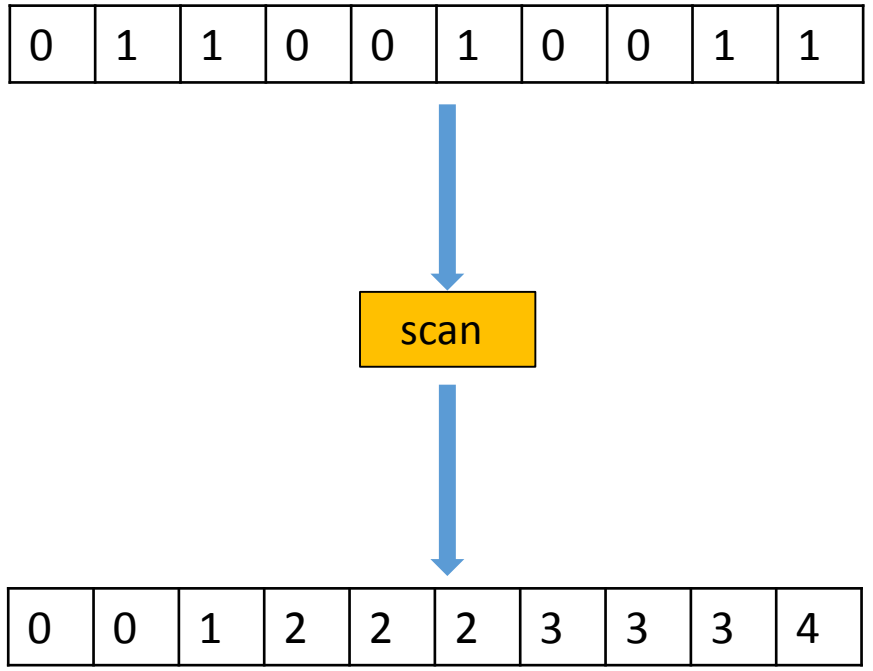
or

```
#define KEEP 1
```

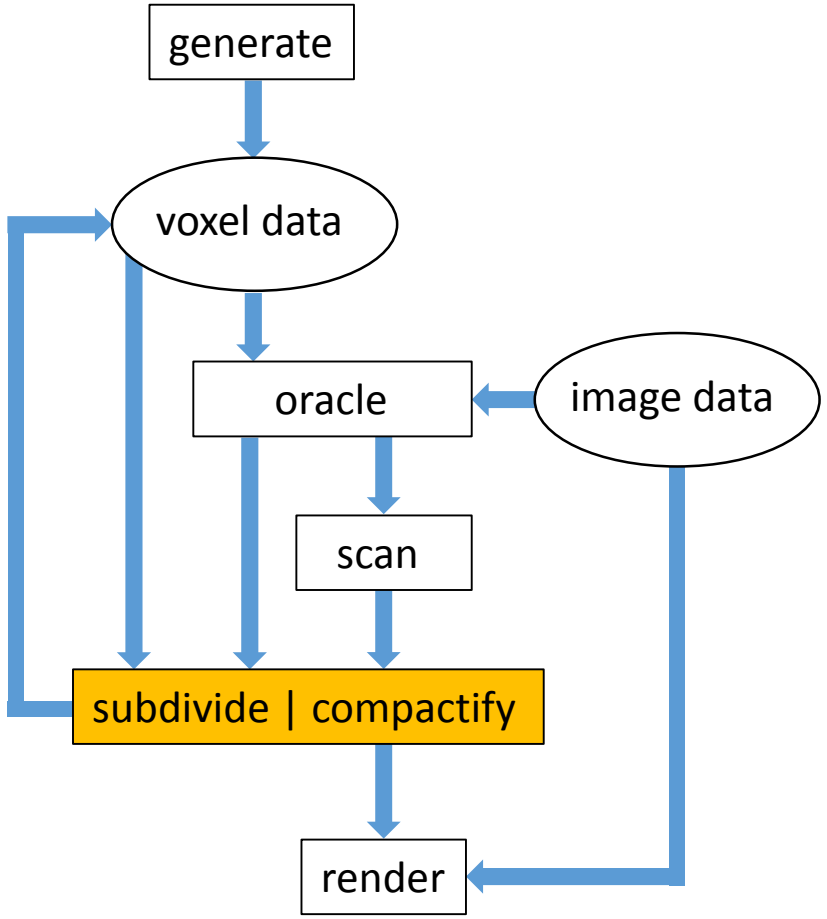
Scan



Exclusive pre-fix sum scan

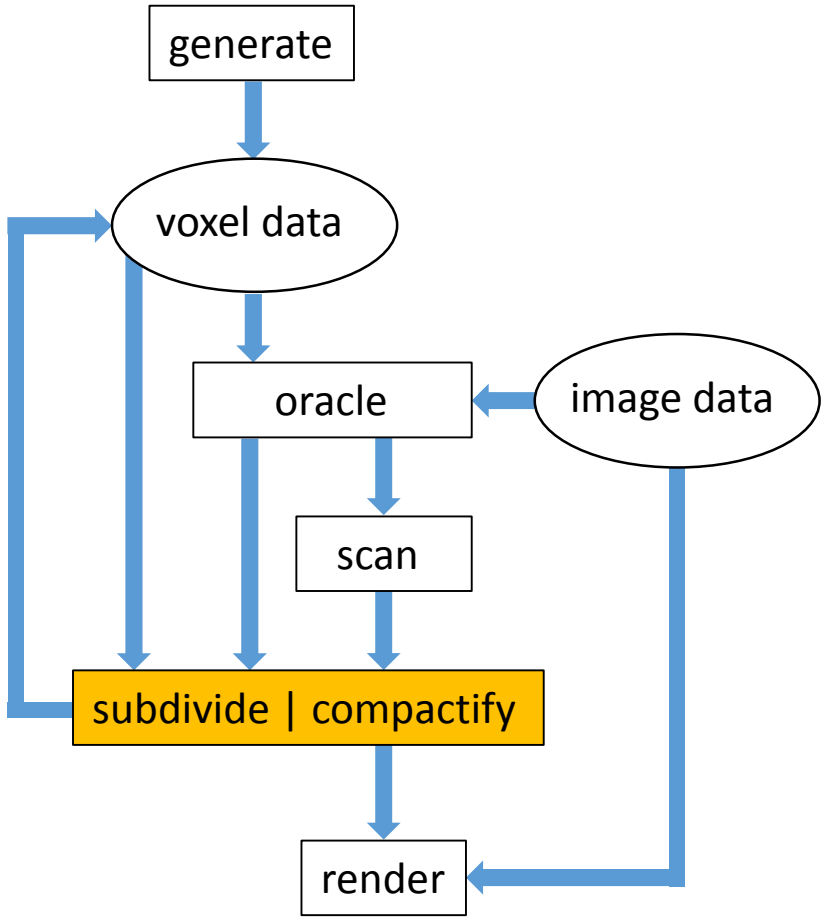


Subdivide



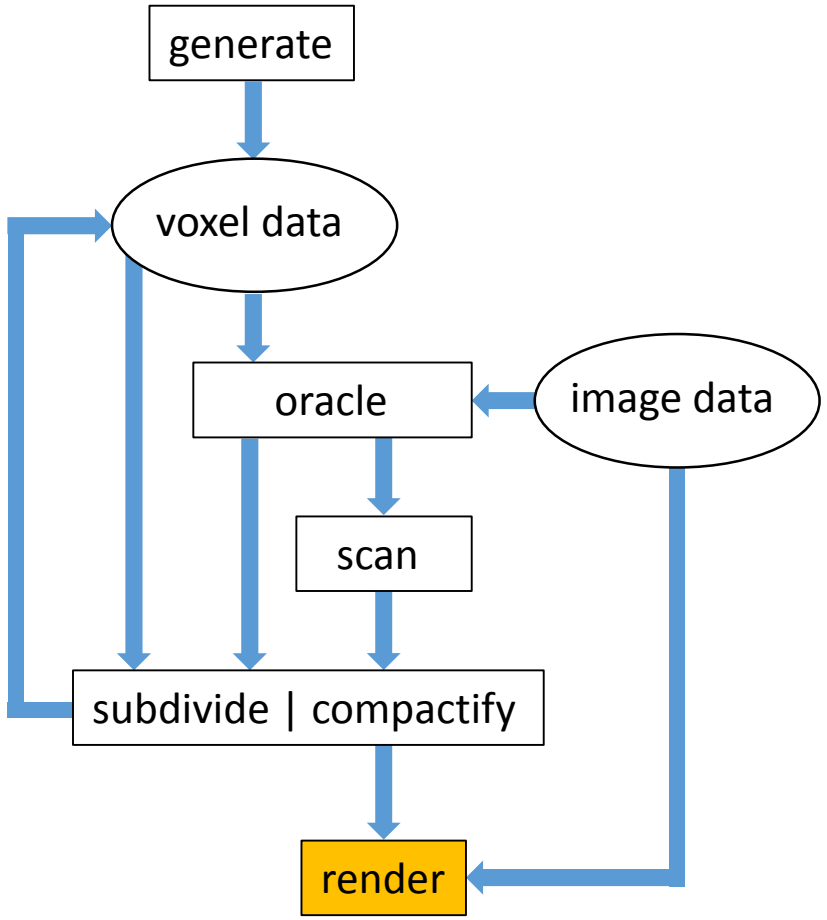

```
if (oracleOutput[threadID] != 0)
 {
     voxel vIn = VoxelListIn[threadID];
     uint baseIdx = 8*scanOutput[threadID];
     for (uint i = 0; i < 8; i++)
     {
          voxel vOut;
          vOut.ix = 2*vIn.ix + i/4;
          vOut.iy = 2*vIn.iy + (i/2)%2;
          vOut.iz = 2*vIn.iz + i%2;
          VoxelListOut[baseIdx+i] = vOut;
     }
 }
```

Compactify




```
if (oracleOutput[threadID] != 0)
 {
     voxel vOut = VoxelListIn[threadID];
     uint baseIdx = scanOutput[threadID];
     VoxelListOut[baseIdx] = vOut;
 }
```

Render



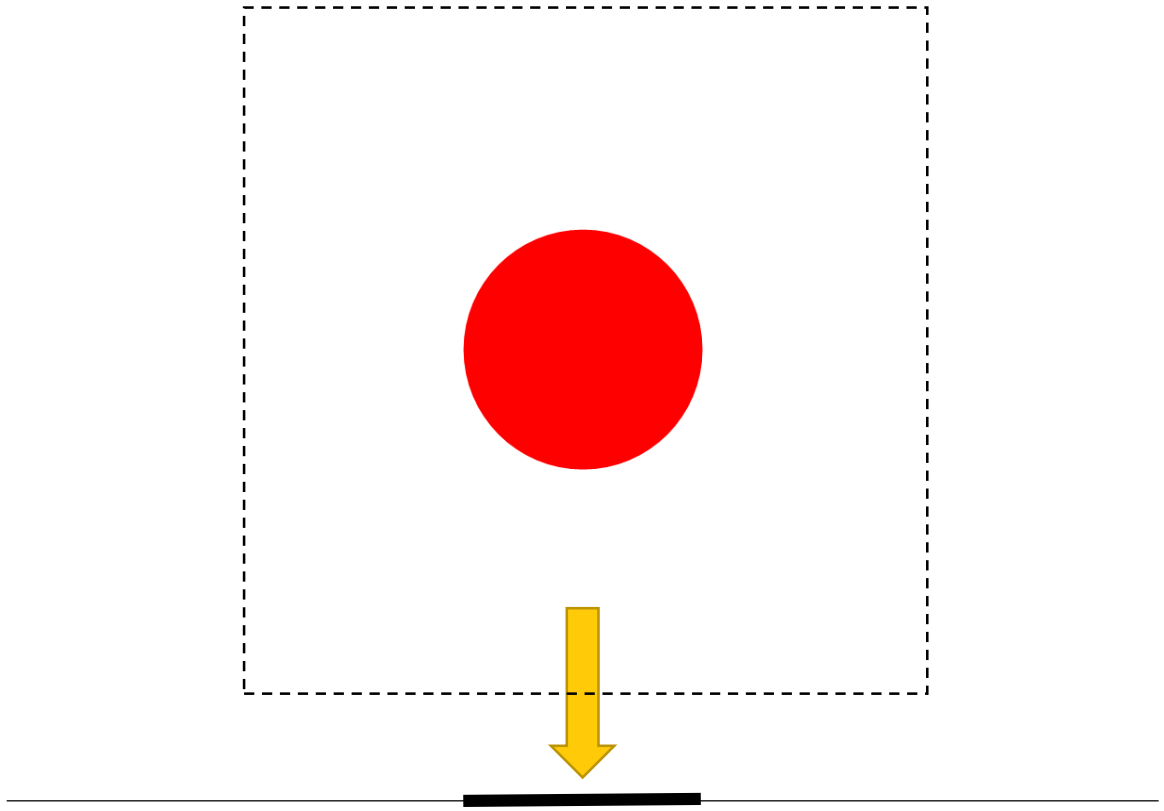
Number of iterations is pre-determined so that voxels are pixel size

Render pixel sized quad - no ray tracing

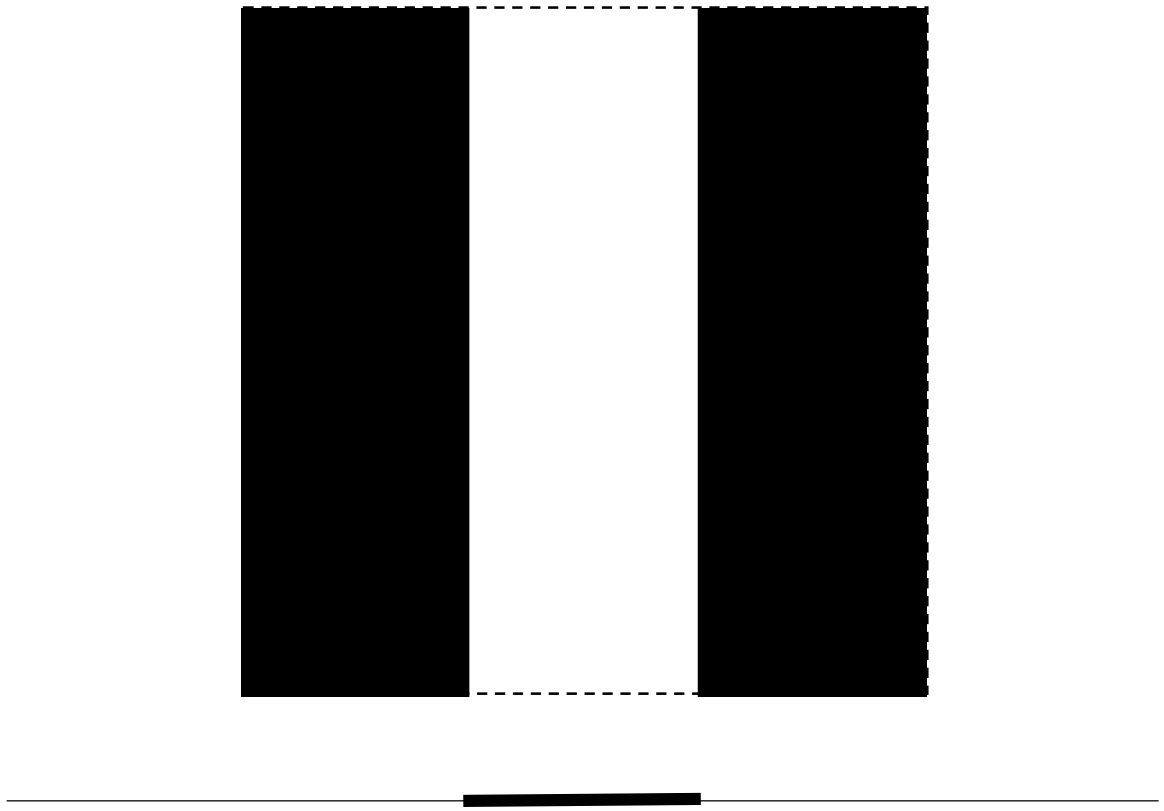
View dependent shading using weighted combination of images

Smooth Visual Hull Reconstruction

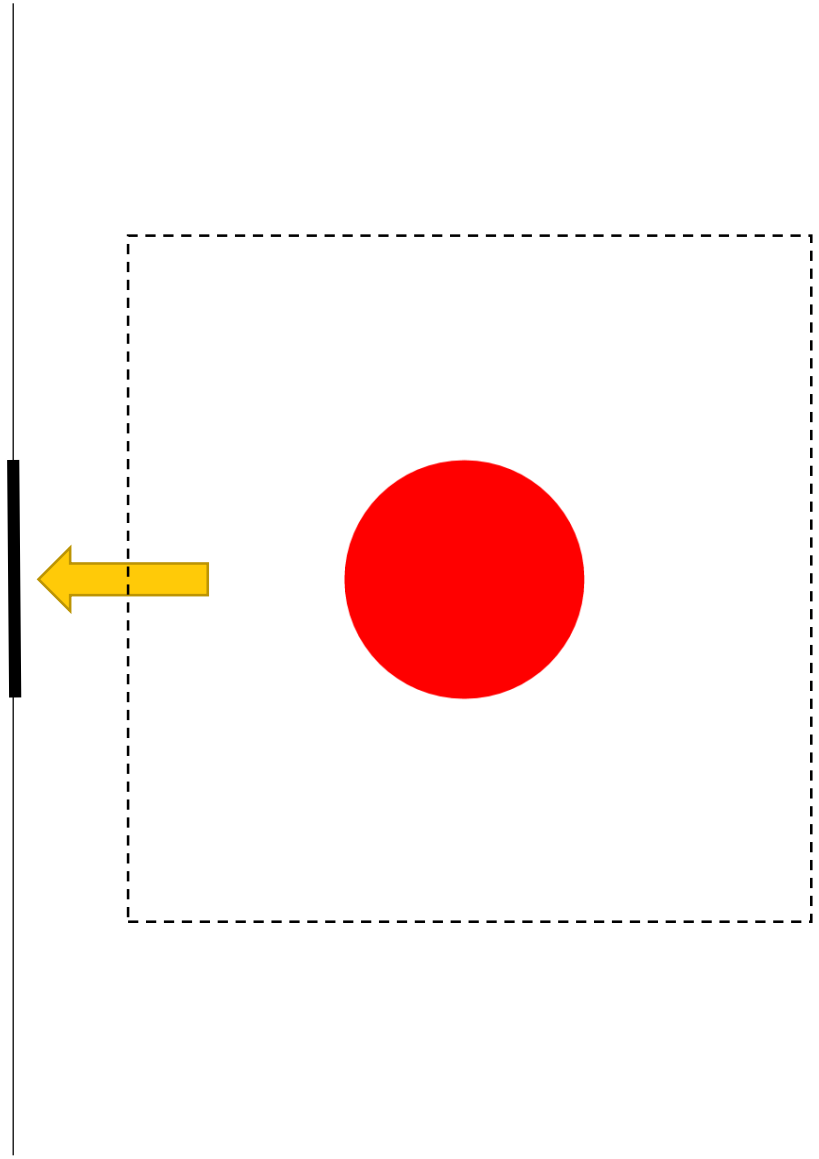
2D Example



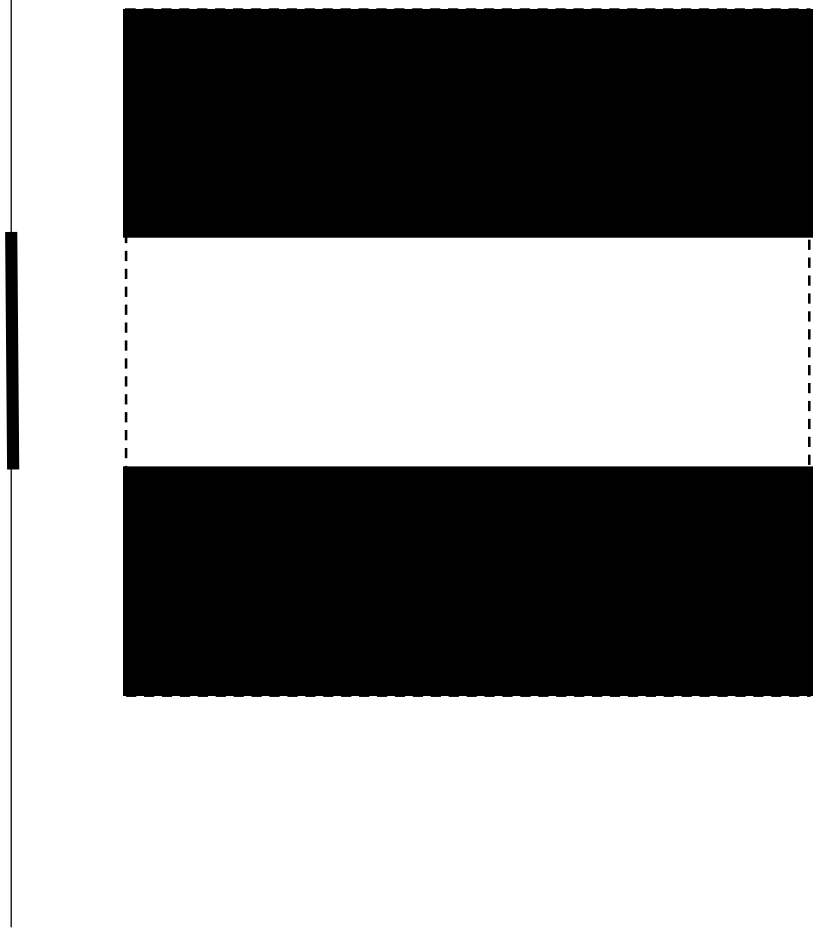
2D Example



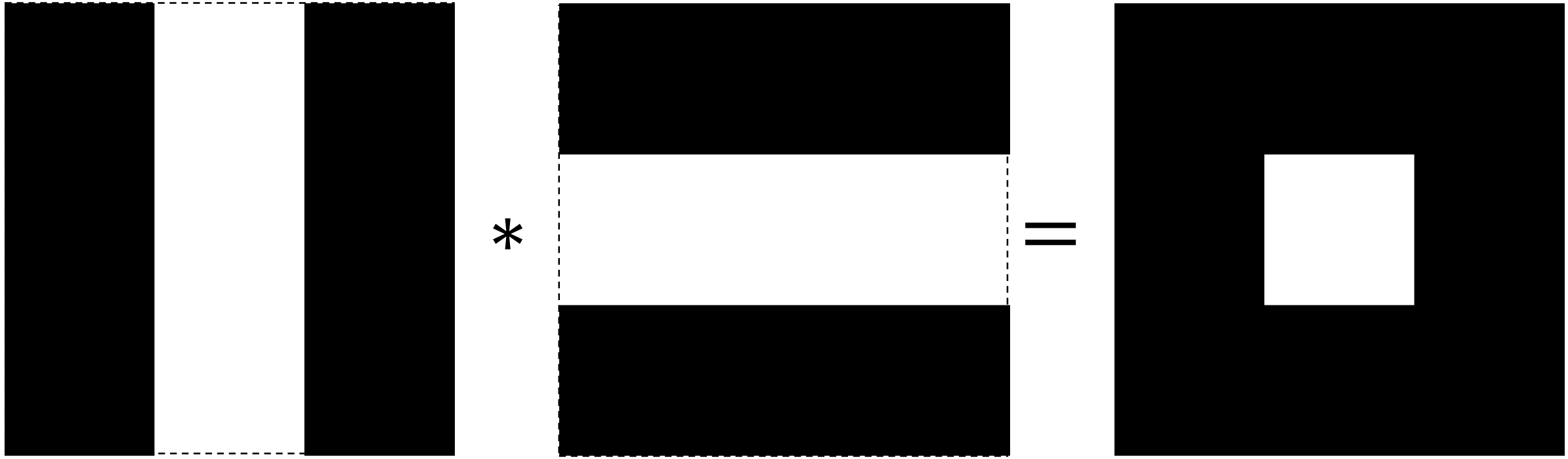
2D Example



2D Example



2D Example



2D Example

Iso-surface: Product of Silhouette Images

$$\boldsymbol{\mathcal{I}}_i : \mathbb{R}^2 \rightarrow [0,1] \qquad \boldsymbol{\mathcal{C}}_i : \mathbb{R}^3 \rightarrow \mathbb{R}^2 \qquad i = 0, \ldots, numCameras$$

$$\boldsymbol{\mathcal{S}}(\mathbf{p}) = \prod_{\boldsymbol{i}} \boldsymbol{\mathcal{I}}_{\boldsymbol{i}}(\boldsymbol{\mathcal{C}}_{\boldsymbol{i}}(\mathbf{p})) \qquad \mathbf{p} \in \mathbb{R}^3$$

Iso-surface: Product of Silhouette Images

$\boldsymbol{\mathcal{I}}_i : \mathbb{R}^2 \rightarrow [0,1]$ $\qquad$ $\boldsymbol{\mathcal{C}}_i : \mathbb{R}^3 \rightarrow \mathbb{R}^2$

$$\boldsymbol{\mathcal{S}}(\mathbf{p}) = \frac{\mathbf{1}}{\mathbf{2}} \qquad \mathbf{p} \in \mathbb{R}^3$$

surface normal: $\nabla \boldsymbol{\mathcal{S}}(\mathbf{p})$

Input Data and Smoothed Silhouette

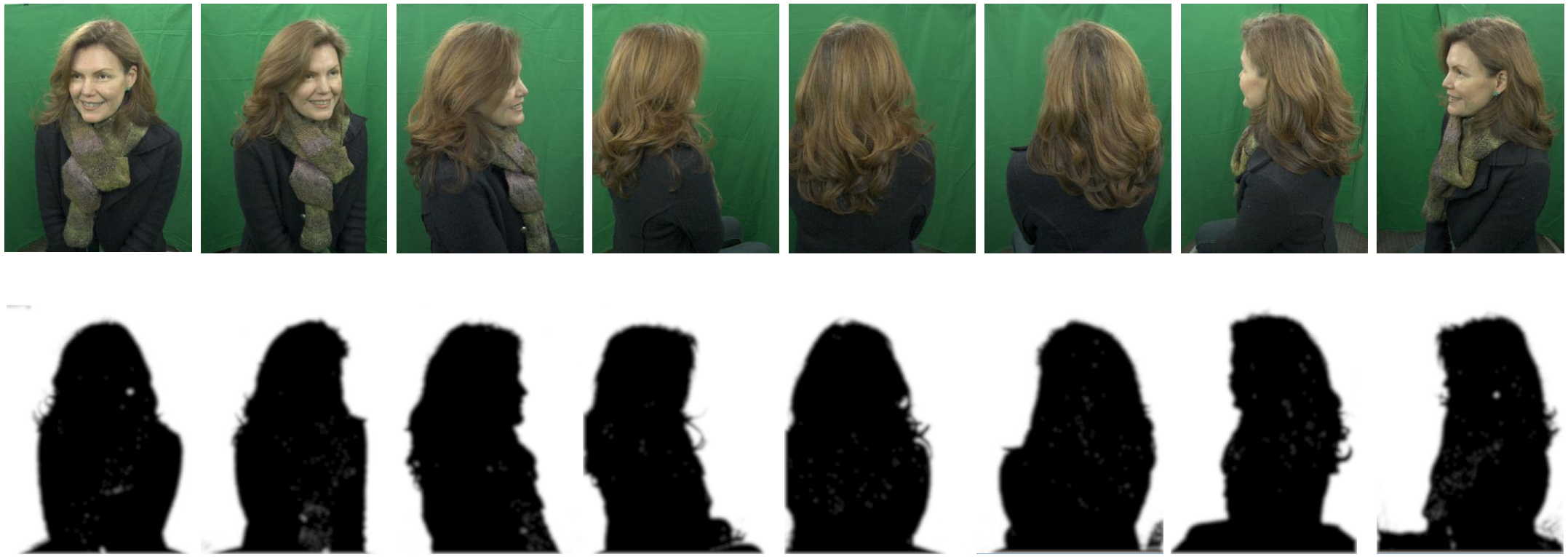
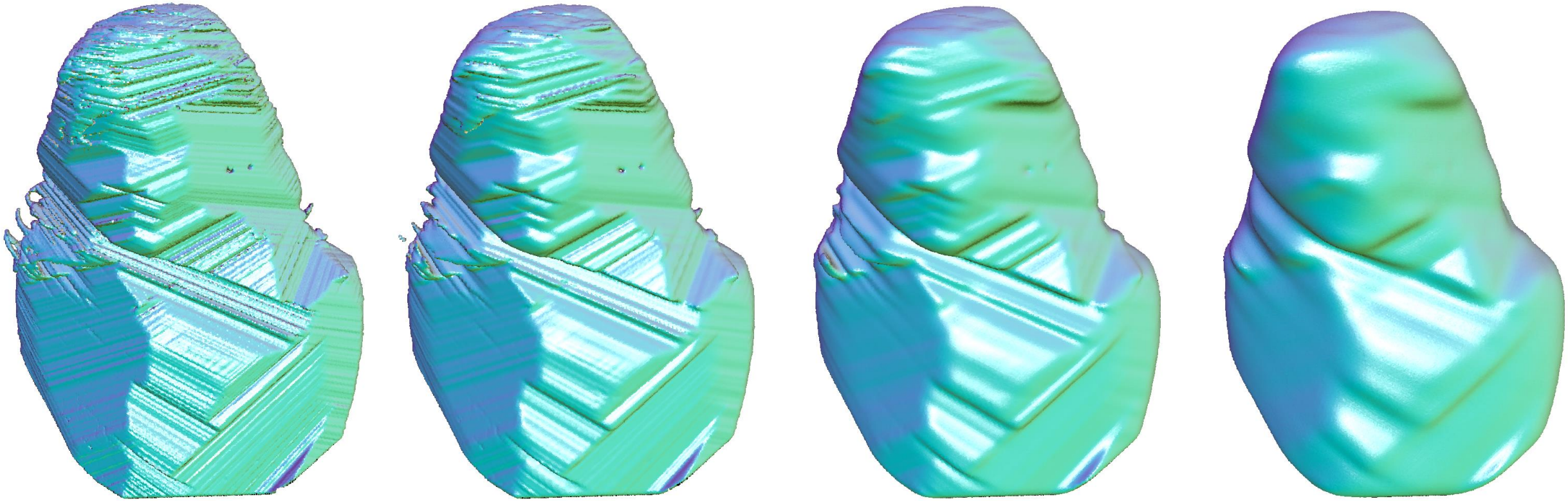


Image size: 800x600

Various Smoothing Kernel Widths



11 21 41 81

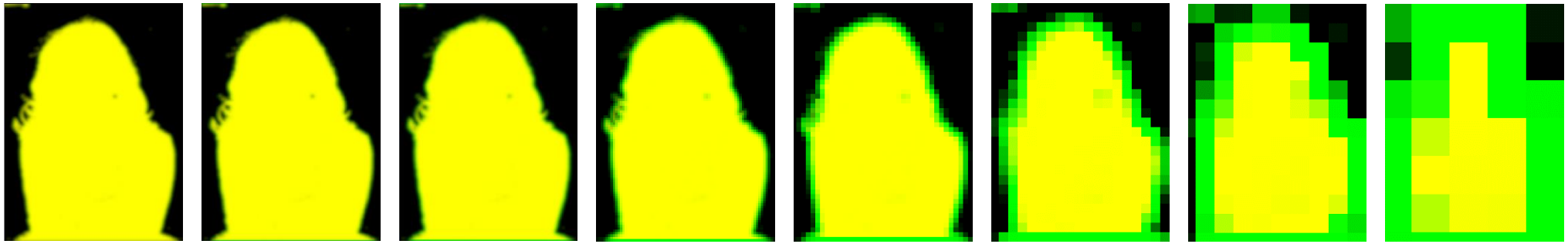
Image processing steps

1) Upload raw camera data
   - Remove radial lens distortion, demosaic, and chroma key
   - Binary silhouette images (0 or 1)

2) Filter
   - Separable unit integral Gaussian kernel of finite width
   - Image row (column) per compute kernel block
   - Creates blur $\in [0,1]$

`sampleSilhouette(v,i)`

- Return bounds of projection of `v` into silhouette image `i`
- Acceleration structure inspired by well known *Hierarchical z-buffer*
- 2 channel mip maps -propagate min and max values up tree



- Project voxel into image space, find AABB: `{minX, minY, maxX, maxY}`
- Choose mip level according to `floor(log2(max(maxX-minX, maxY-minY)))`
- Return min and max over 2x2 texels `float2 bound;`

Oracle Kernel

```
DECISION = CULL;
Voxel v = VoxelList[threadID];

If (!outsideFrustum(v))
{
    bounds = float2(1.0, 1.0);
    for (int i=0; i < numCameras; i++)
        bounds *= sampleSilhouette(v, i);

    if (bounds.x < 0.5 && bounds.y > 0.5)
        DECISION = KEEP;
}

oracleOutput[threadID] = DECISION;
```

Results and Conclusions

Results

Input images (8)
8 bits per pixel
800x600

Output image
1920x1080

	GeForce Titan		Radeon 7970GE	
level `k`	9	10	9	10
demosaic, etc.	0.46	0.46	0.43	0.43
convolution	2.49	2.49	1.56	1.56
mip generation	0.44	0.44	0.58	0.58
generate	0.06	0.06	0.11	0.11
oracle	1.66	5.27	2.09	6.47
scan	1.48	3.19	1.01	2.69
subdivide	0.87	2.33	0.98	2.71
compactify	0.23	0.71	0.21	0.61
compute normals	0.49	1.74	0.56	1.97
render	1.57	6.30	4.47	17.87
Total	9.75	23.86	14.64	35.00
fps	112	46	83	30

Level	9	10
numVoxels	606660	2538713

Conclusions

- Sparse, high- resolution voxel technique is highly effective
- Smooth visual hull concept great for hair
- Bottleneck
  - Data throughput – from cameras to GPU memory

Thank you