Directly Visualizing Volume Data

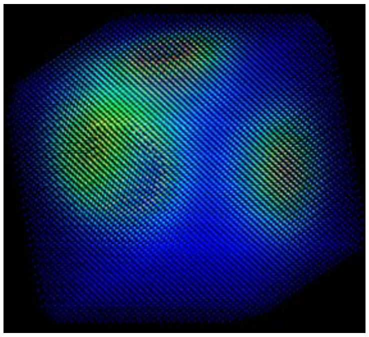
Mike Bailey

mjb@cs.oregonstate.edu

Oregon State University

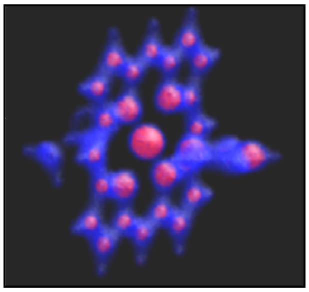
Volume Data: A Definition

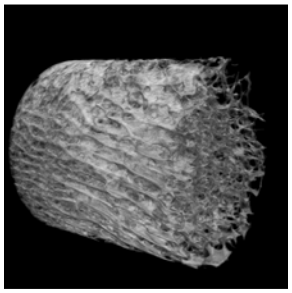
A *volume* is a 3D discretely sampled data set where the size of the voxels have been expanded to occupy the space to the neighboring voxels.



Why Do We Care About Volume Visualization?

- Medical: CAT, MRI, 3D ultrasound
- Science and engineering: CFD, stress, thermal, molecular
- Volumes are normally very difficult to comprehend





How can you get a volume dataset? (Ewww…)

Montreal Neurological Institute at McGill University

Researchers used a tool called a microtome to cut a brain into slices 20 micrometers thick.

Understanding Volume Data Usually Involves a Compromise

Point Clouds ➡ All values everywhere, hard to see very much, distracting artifacts

Interpolated-colors cutting planes ➡ All values in a single plane

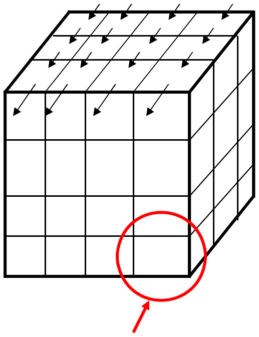
Contours cutting plane ➡ Discrete values in a single plane

Isosurfaces ➡ A single value everywhere

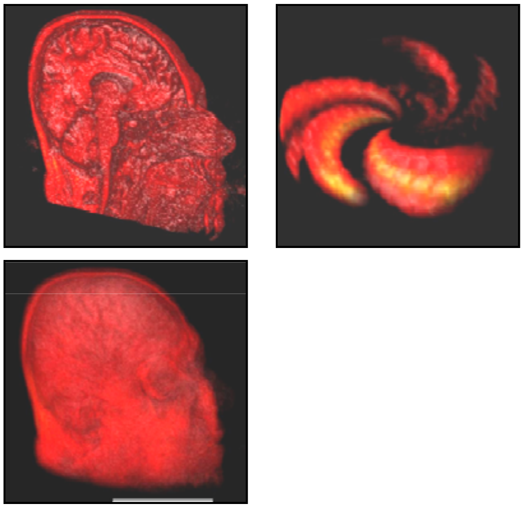
Because of these compromises, these are all considered to be ***indirect*** ways to visualize volume data

Direct Volume Rendering

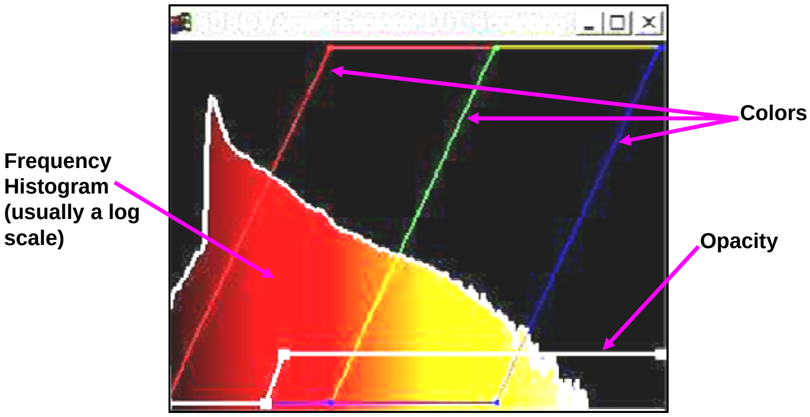
Composite the colors and alphas of the voxels



A *Volume Element*, or *voxel*



Transfer Function



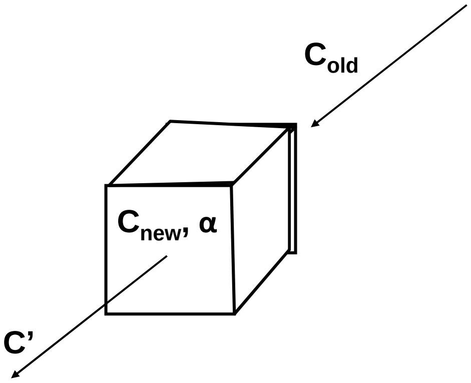
OSU vx Transfer Function Sculpting Window

Voxel Compositing

Recall this color blending equation from the OpenGL Transparency notes:

$$C' = \alpha C_{new} + (1. - \alpha) C_{old}$$

In “Voxel World”, things work the same way:

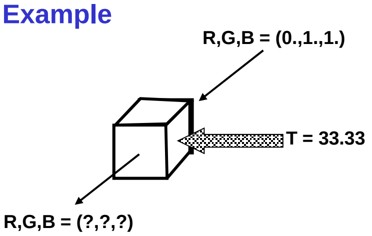


Voxel Compositing Example

TMIN = 0.
TMAX = 100.

The color transfer function is a **Black-Red-Yellow-White heated object scale**, mapping a scalar value of 0. to Black, and 100. to White.

The opacity transfer function is a linear ramp so that the opacity is 1. (opaque) when T = 100. and 0. (transparent) when T = 0.



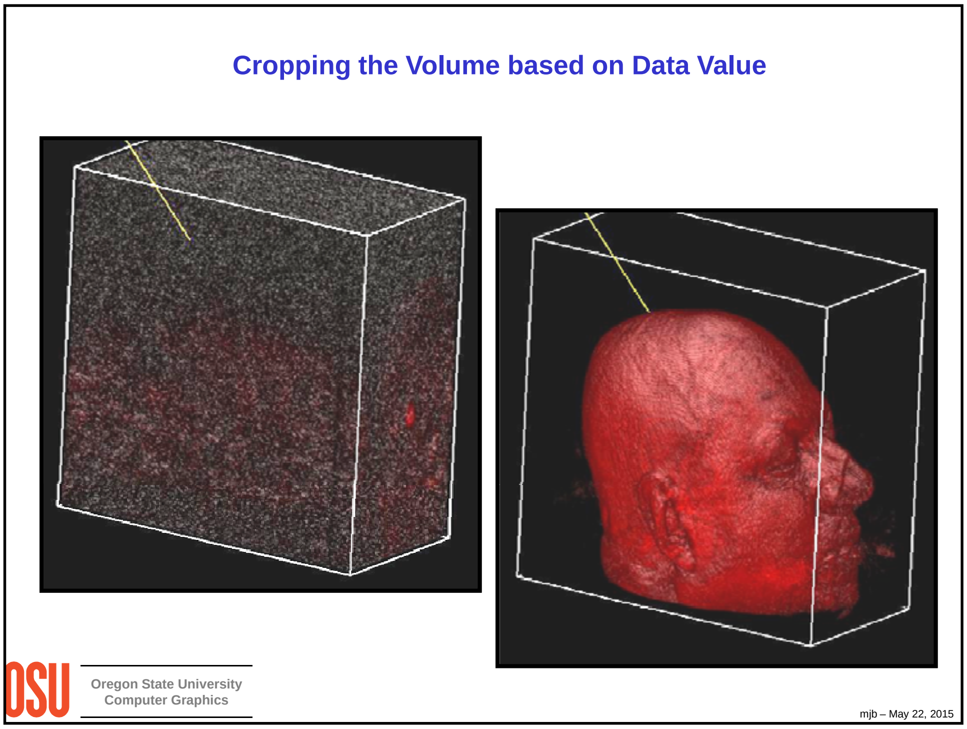
You are compositing back-to-front through the volume. At this moment, the running values of RGB are **(0., 1., 1.) .** The next voxel you encounter has a **T value of 33.33**

1. **What is the color of *just this voxel*?**
2. **What is the opacity of *just this voxel*?**
3. **What will the new running RGB values be when you are done compositing this voxel with the old running RGB values?**

What is the color of *just this voxel*?

What is the opacity of *just this voxel*?

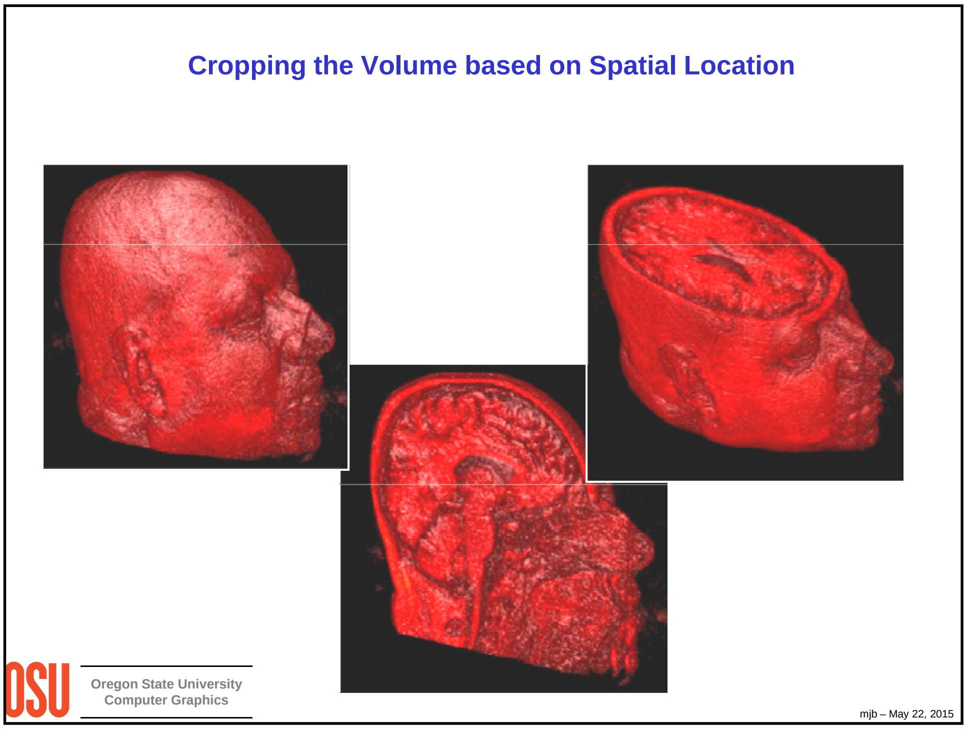
What will the new running RGB values be when you are done compositing this voxel with the old running RGB values?



Cropping the Volume based on Data Value

Oregon State University
Computer Graphics

mjb – May 22, 2015

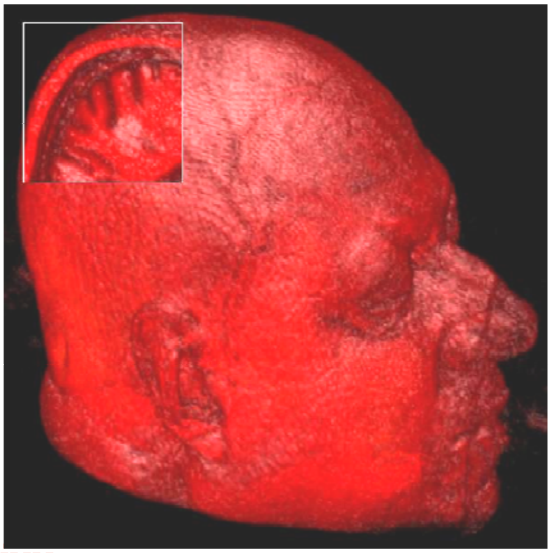


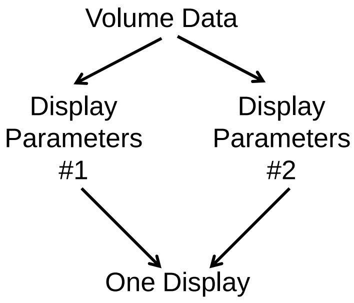
Cropping the Volume based on Spatial Location

Oregon State University
Computer Graphics

mjb – May 22, 2015

“Magic Lens” to Selectively Look Inside





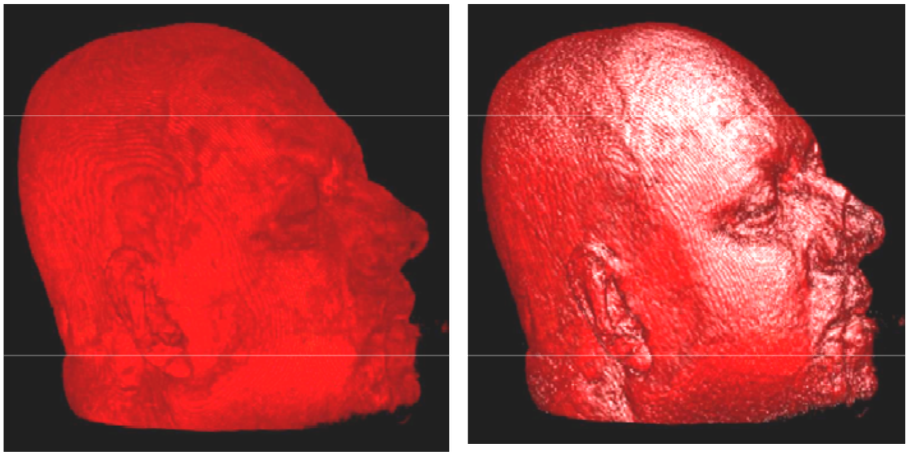
Volume Data

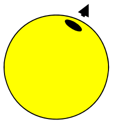
Display Parameters #1

Display Parameters #2

One Display

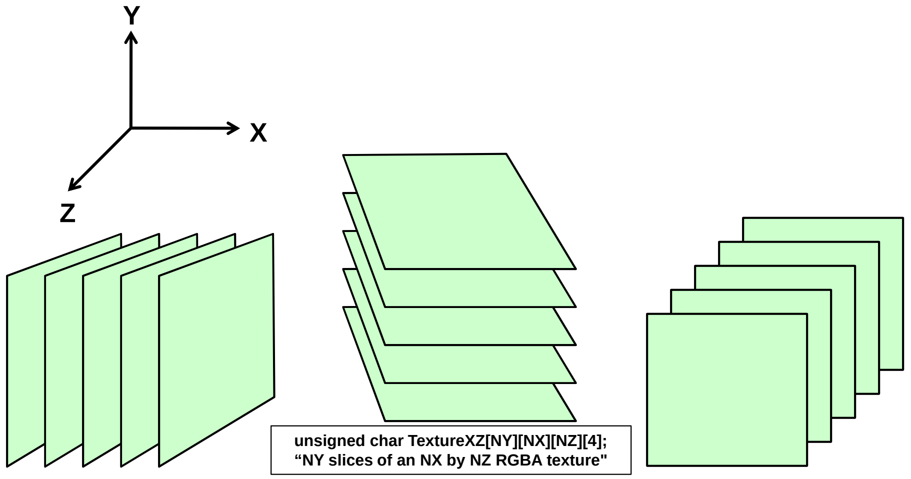
Lighting





$$\vec{n} = (\frac{dS}{dx}, \frac{dS}{dy}, \frac{dS}{dz}) = \nabla S$$

Volume Rendering with Parallel Texture Planes



unsigned char TextureXZ[NY][NX][NZ][4];
“NY slices of an NX by NZ RGBA texture"

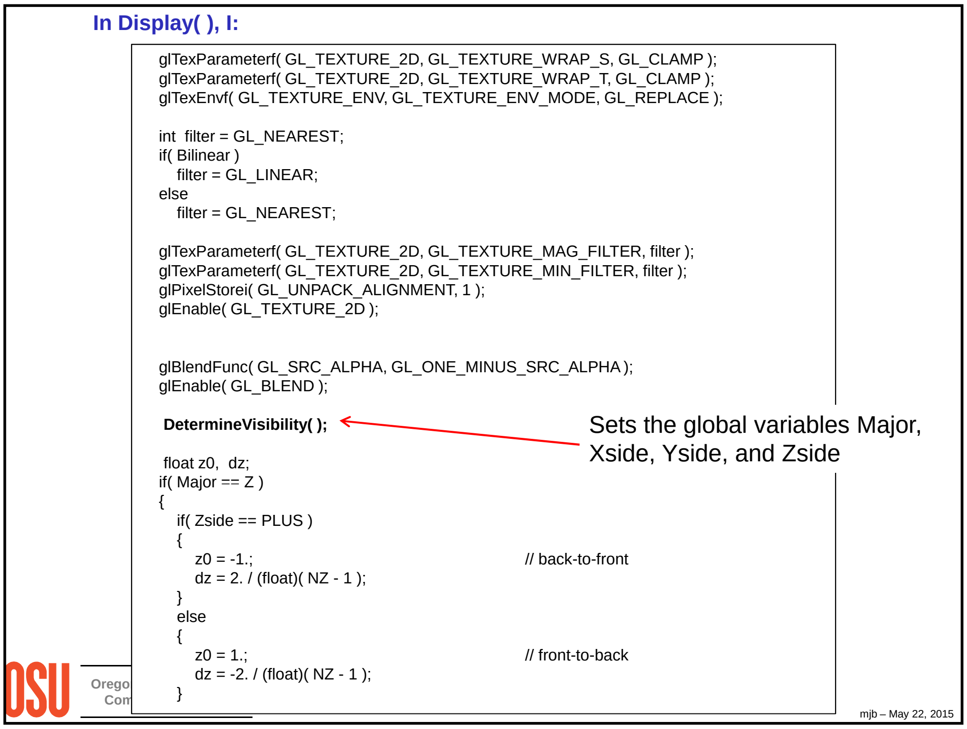
unsigned char TextureYZ[NX][NY][NZ][4];
“NX slices of an NY by NZ RGBA texture"

unsigned char TextureXY[NZ][NX][NY][4];
“NZ slices of an NX by NY RGBA exture"

In a callback that is called whenever the opacity transfer function changes:

```
void
FillXY( void )
{
    float alpha;        // opacity at this voxel
    float r, g, b;      // running color composite

    for( int x = 0; x < NX; x++ )
    {
        for( int y = 0; y < NY; y++ )
        {
            r = g = b = 0.;
            for( int zz = 0; zz < NZ; zz++ )
            {
                // which direction to fill:

                int z;
                if( Zside == PLUS )
                    z = zz;
                else
                    z = ( NZ-1 ) - zz;

                if(  … this scalar value is not in the range you want to view …  )
                {
                    r = g = b = 0.;
                    alpha = 0.;
                }
                else
                {
                    r = Nodes[x][y][z].r;
                    g = Nodes[x][y][z].g;
                    b = Nodes[x][y][z].b;
                    alpha = MaxAlpha;
                }

                TextureXY[zz][y][x][0] = (unsigned char) ( 255.*r + .5 );
                TextureXY[zz][y][x][1] = (unsigned char) ( 255.*g + .5 );
                TextureXY[zz][y][x][2] = (unsigned char) ( 255.*b + .5 );
                TextureXY[zz][y][x][3] = (unsigned char) ( 255.*alpha + .5 );
            }
        }
    }
}
```

Zside is set from somewhere else

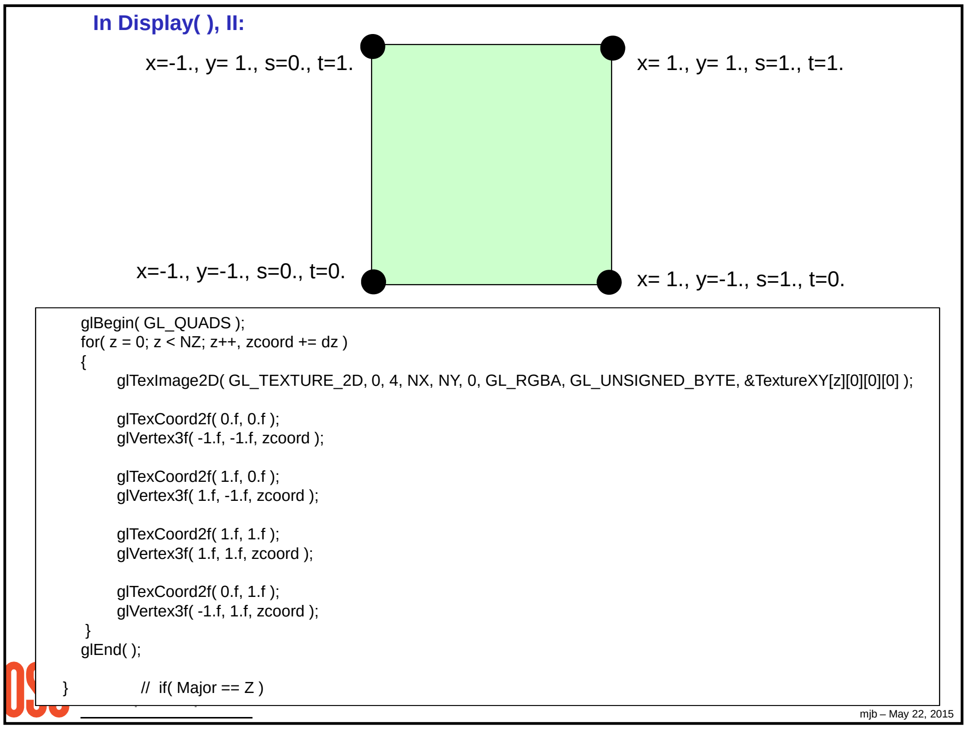


In Display( ), I:

```
glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP );
glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP );
glTexEnvf( GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE );

int  filter = GL_NEAREST;
if( Bilinear )
    filter = GL_LINEAR;
else
    filter = GL_NEAREST;

glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, filter );
glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, filter );
glPixelStorei( GL_UNPACK_ALIGNMENT, 1 );
glEnable( GL_TEXTURE_2D );


glBlendFunc( GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA );
glEnable( GL_BLEND );

 DetermineVisibility( );

 float z0,  dz;
if( Major == Z )
{
    if( Zside == PLUS )
    {
        z0 = -1.;                                   // back-to-front
        dz = 2. / (float)( NZ - 1 );
    }
    else
    {
        z0 = 1.;                                    // front-to-back
        dz = -2. / (float)( NZ - 1 );
    }
```

Sets the global variables Major, Xside, Yside, and Zside

mjb – May 22, 2015



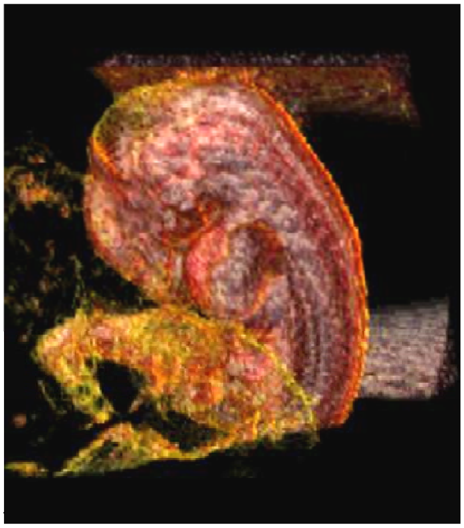
In Display( ), II:

x=-1., y= 1., s=0., t=1.
x= 1., y= 1., s=1., t=1.
x=-1., y=-1., s=0., t=0.
x= 1., y=-1., s=1., t=0.

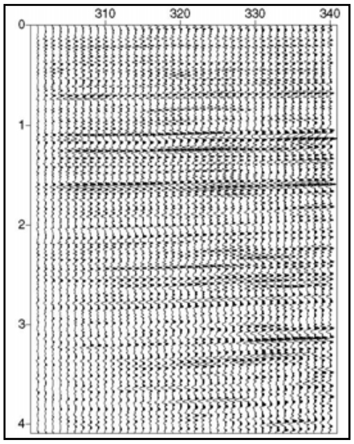
```
     glBegin( GL_QUADS );
     for( z = 0; z < NZ; z++, zcoord += dz )
     {
          glTexImage2D( GL_TEXTURE_2D, 0, 4, NX, NY, 0, GL_RGBA, GL_UNSIGNED_BYTE, &TextureXY[z][0][0][0] );

          glTexCoord2f( 0.f, 0.f );
          glVertex3f( -1.f, -1.f, zcoord );

          glTexCoord2f( 1.f, 0.f );
          glVertex3f( 1.f, -1.f, zcoord );

          glTexCoord2f( 1.f, 1.f );
          glVertex3f( 1.f, 1.f, zcoord );

          glTexCoord2f( 0.f, 1.f );
          glVertex3f( -1.f, 1.f, zcoord );
      }
     glEnd( );

  }            //  if( Major == Z )
```

mjb – May 22, 2015

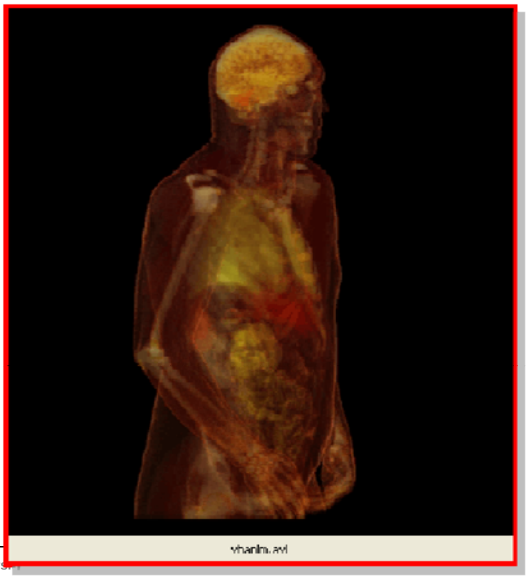
Human Embryo



Geophysics

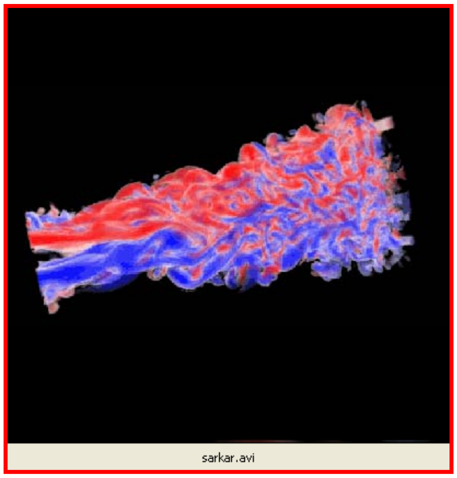


Volume Interaction: The Visible Human

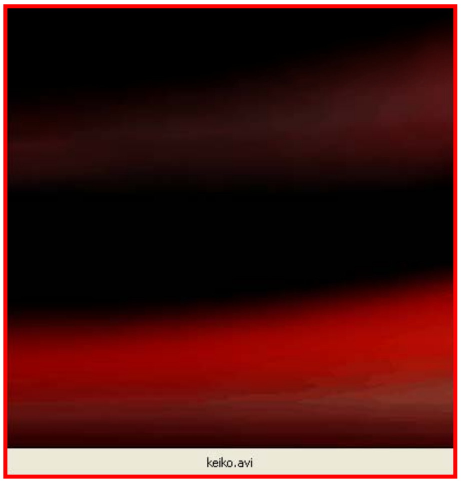


vhanim.avi

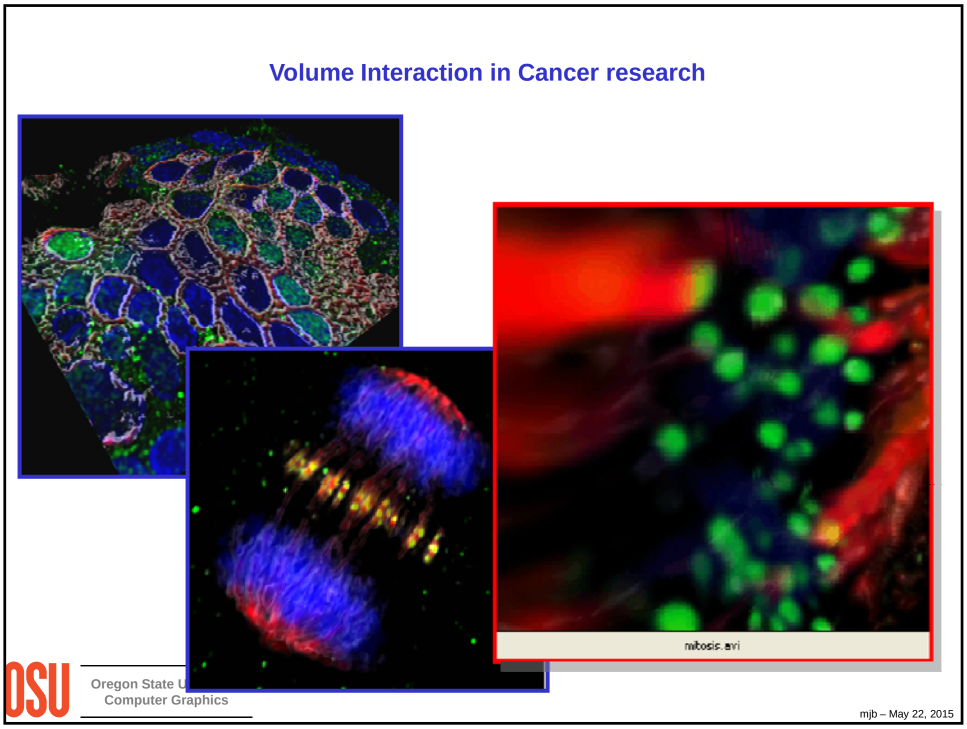
Interactive Volume Visualization for Computational Fluid Dynamics



sarkar.avi



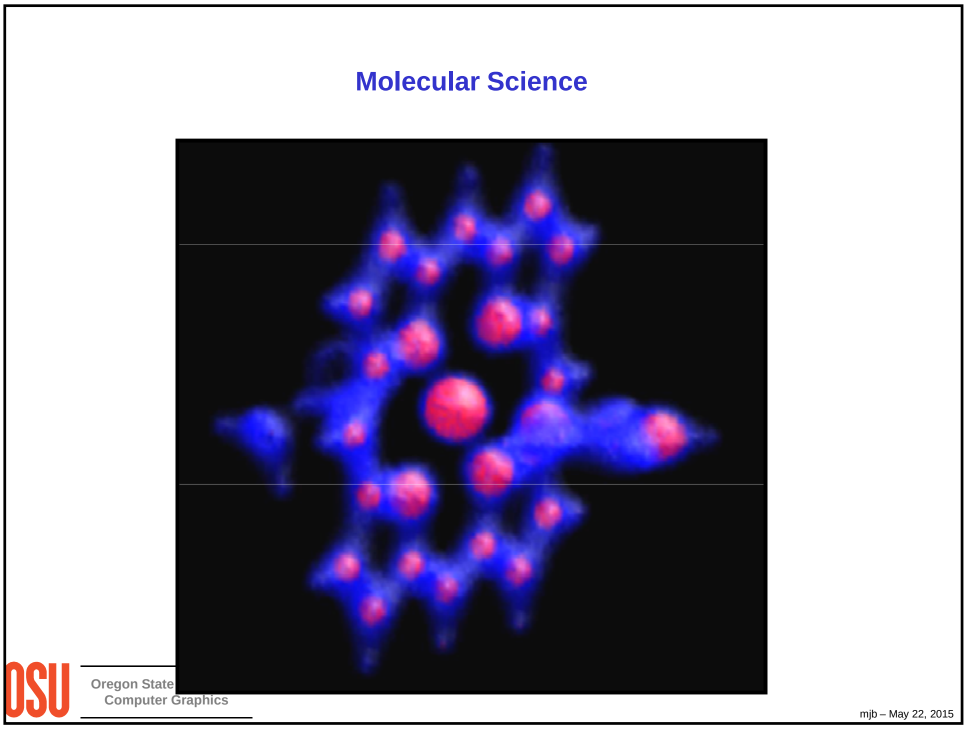
keiko.avi



Volume Interaction in Cancer research

Oregon State U
Computer Graphics

mjb – May 22, 2015

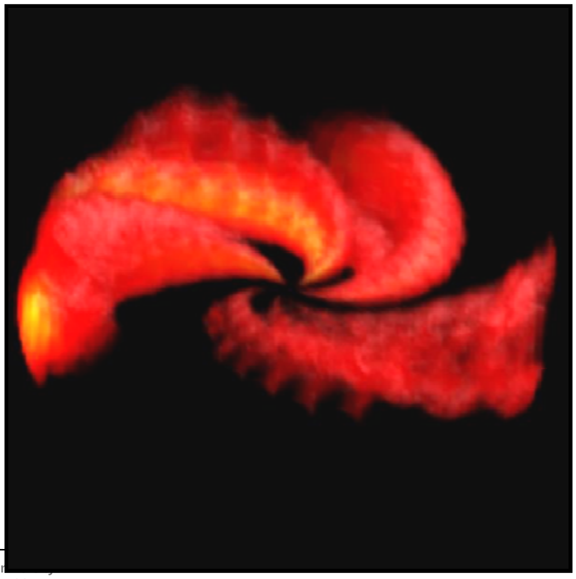


Molecular Science

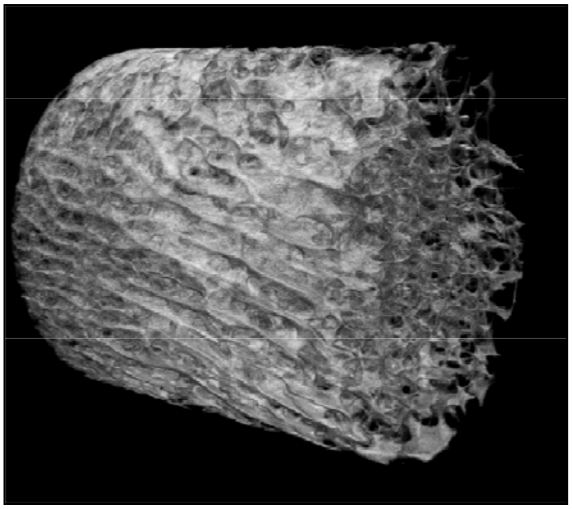
Oregon State
Computer Graphics

mjb – May 22, 2015

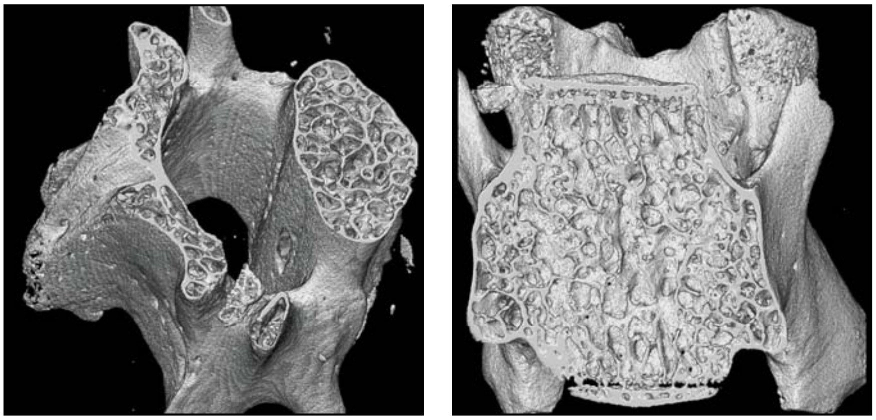
Solar Wind



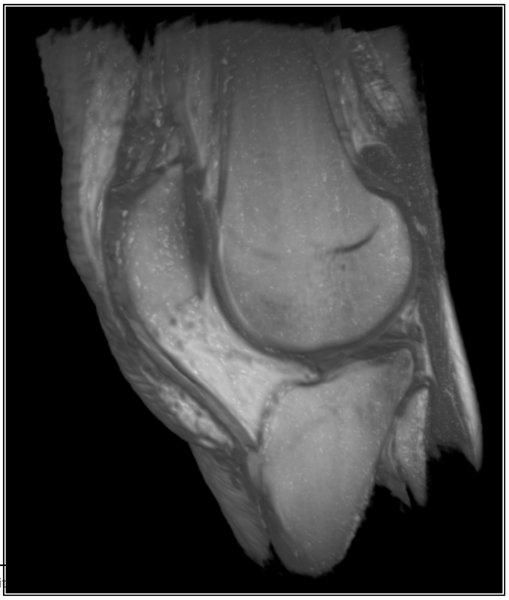
OSU Sheepbone



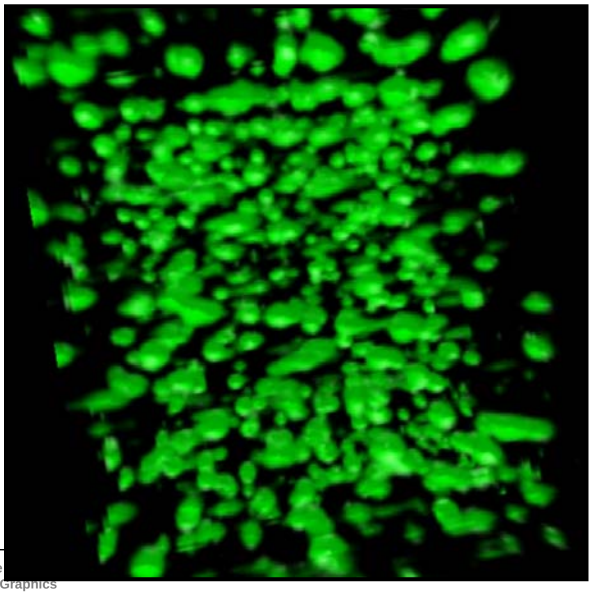
OSU Mouse Vertrebra



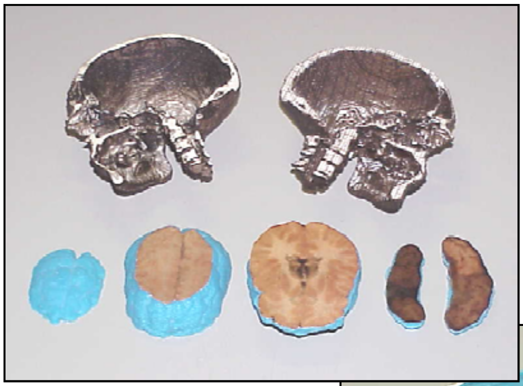
Professor Metoyer’s Knee



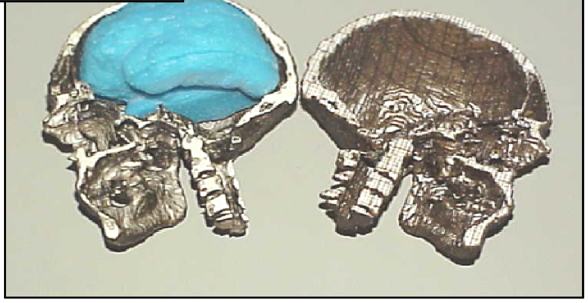
Foliage Density



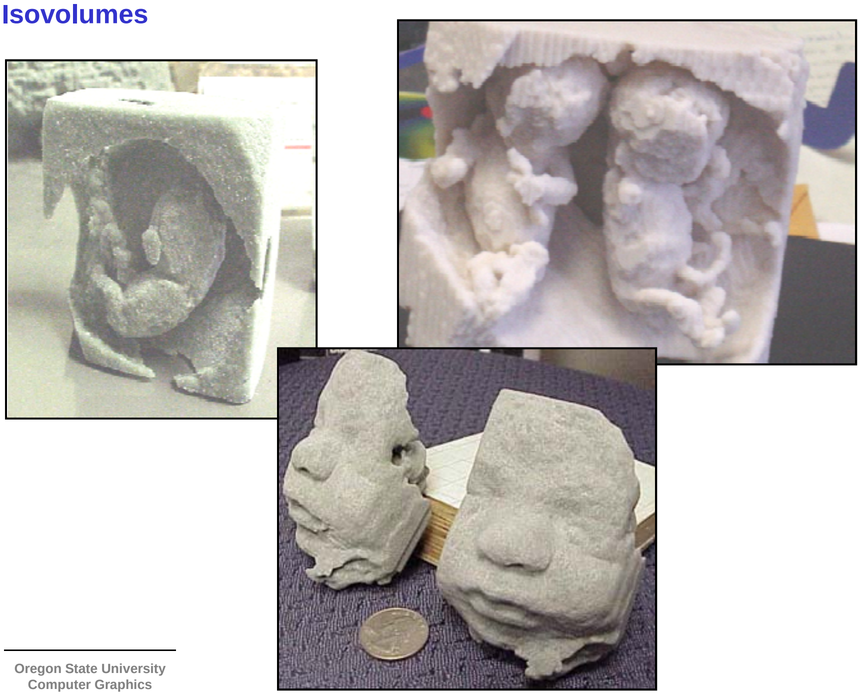
Isovolumes



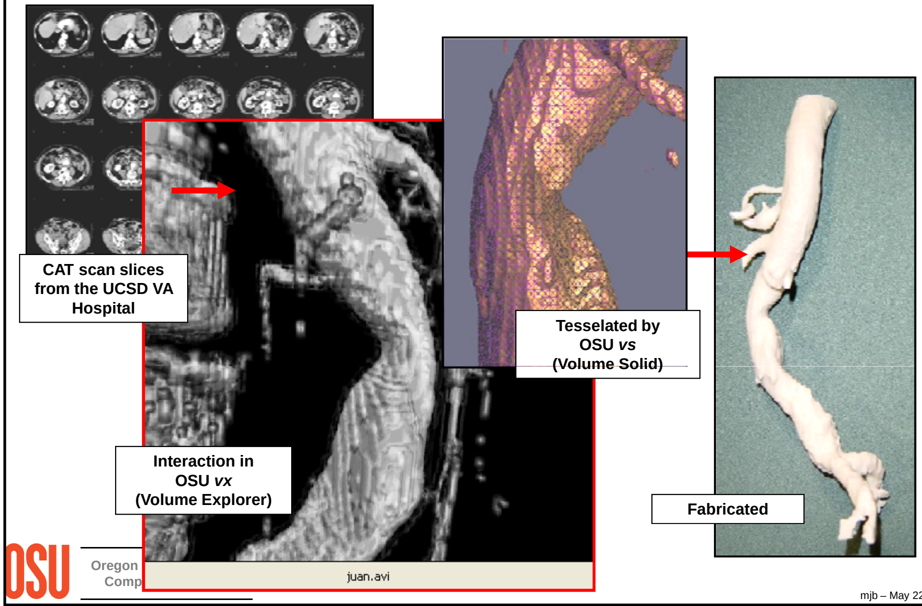
To be manufactureable, there must be finite material between two isosurfaces



Isovolumes



Putting the Tools Together:
Modeling and Making Anabolic Aortic Aneurysms



CAT scan slices from the UCSD VA Hospital

Interaction in OSU *vx* (Volume Explorer)

Tesselated by OSU *vs* (Volume Solid)

Fabricated