Regular OH

- Tues 10-12 and Weds 2-4
- Surge OH on Fri’s, esp when Tues OH are cancelled
- Tues OH cancelled on Tues, May 14
GPU Acceleration of Particle Advection Workloads in a Parallel, Distributed Memory Setting

David Camp, Hari Krishnan, David Pugmire, Christoph Garth, Ian Johnson, E. Wes Bethel, Kenneth I. Joy, and Hank Childs

April 25, 2013  Hank Childs, Univ. of Oregon / LBNL
Clear what to do?

Clear what to hand in?

Questions?

(don’t send me your movies)
Making movies

- **Basic idea:**
  - Program A: output a bunch of images
    - Program A == Project 1F
  - Program B: collect images and encode them into a movie format.
    - Program B == ffmpeg, mpeg_encode, etc

- **My ffmpeg invocation:**
  - `ffmpeg -f image2 -i frame%d.png -vcodec mpeg4 -mbd rd -r 30 -b 18000000 movie.mp4`

- Easy to find this stuff on the web…
- Share successes on Piazza..
How to make a graphics program?

- Need to create a window
  - This window contains a “context” for OpenGL to render in.
- Need to be able to deal with events/interactions
- Need to render graphics primitives
  - OpenGL!
Creating windows and dealing with events varies from platform to platform.
“Hello World”
with X-Windows.

Compile with:

```
gcc -L/usr/X11R6/lib -lX11 hello-x.c -o hello-x
```

```c
#include <X11/Xlib.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void) {
    Display *d;
    Window w;
    XEvent e;
    char *msg = "Hello, World!";
    int s;

    d = XOpenDisplay(NULL);
    if (d == NULL) {
        fprintf(stderr, "Cannot open display\n");
        exit(1);
    }

    s = DefaultScreen(d);
    w = XCreateSimpleWindow(d, RootWindow(d, s), 10, 10, 100, 100, 1,
                             BlackPixel(d, s), WhitePixel(d, s));
    XSelectInput(d, w, ExposureMask | KeyPressMask);
    XMapWindow(d, w);

    while (1) {
        XNEvt(d, &e);
        if (e.type == Expose) {
            XFillRectangle(d, w, DefaultGC(d, s), 20, 20, 10, 10);
            XDrawString(d, w, DefaultGC(d, s), 10, 50, msg, strlen(msg));
        }
        if (e.type == KeyPress)
            break;
    }

    XCloseDisplay(d);
    return 0;
}
```
Creating windows and dealing with events varies from platform to platform.

Some packages provide implementations for key platforms (Windows, Unix, Mac) and abstractions for dealing with windows and events.

GLUT: library for cross-platform windowing & events.
- My experiments: doesn’t work as well as it used to.

VTK: library for visualization
- But also contains cross-platform windowing & events.
Visualization with VTK

Content from: Erik Vidholm, Univ of Uppsula, Sweden
David Gobbi, Robarts Research Institute, London, Ontario, Canada
Open source, freely available software for 3D computer graphics, image processing, and visualization

Managed by Kitware Inc.

Use C++, Tcl/Tk, Python, Java
True visualization system

- Visualization techniques for visualizing
  - Scalar fields
  - Vector fields
  - Tensor fields
- Polygon reduction
- Mesh smoothing
- Image processing
- Your own algorithms
Additional features

- Parallel support (message passing, multithreading)
- Stereo support
- Integrates easily with Motif, Qt, Tcl/Tk, Python/Tk, X11, Windows, ...
- Event handling
- 3D widgets
3D graphics

- Surface rendering
- Volume rendering
  - Ray casting
  - Texture mapping (2D)
  - Volume pro support
- Lights and cameras
- Textures
- Save render window to .png, .jpg, ...
  (useful for movie creation)
The visualization pipeline

DATA → FILTER → MAPPING → DISPLAY

Visualization algorithms

Interactive feedback
We will replace these and write our own GL calls.

Cone.py Pipeline Diagram (type "python Cone.py" to run)

Source
- Data
  - Either reads the data from a file or creates the data from scratch.

Mapper
- OpenGL
  - Moves the data from VTK into OpenGL.
  - For setting colors, surface properties, and the position of the object.

Actor
- The rectangle of the computer screen that VTK draws into.
- The window, including title bar and decorations.
  - Allows the mouse to be used to interact with the data.

Renderer

Window

Interactor

from vtkpython import *
cone = vtkConeSource()
cone.SetResolution(10)
coneMapper = vtkPolyDataMapper()
coneMapper.SetInput(cone.GetOutput())
coneActor = vtkActor()
coneActor.SetMapper(coneMapper)
ren = vtkRenderer()
ren.AddActor(coneActor)
renWin = vtkRenderWindow()
renWin.SetWindowName("Cone")
renWin.SetSize(300, 300)
renWin.AddRenderer(ren)
iren = vtkRenderWindowInteractor()
iren.SetRenderWindow(renWin)
iren.Initialize()
iren.Start()
How to make a graphics program?

- Need to create a window
  - This window contains a “context” for OpenGL to render in.
- Need to be able to deal with events/interactions
- Need to render graphics primitives
  - OpenGL!

Borrow | Build
OpenGL Functions

- Primitives
  - Points
  - Line Segments
  - Polygons
- Attributes
- Transformations
  - Viewing
  - Modeling
- Control (VTK)
- Input (VTK)
- Query
First OpenGL programs

- Remember: none of these programs have windowing or events.
- They contain just the code to put primitives on the screen, with lighting and colors.
class vtk441PolyDataMapper : public vtkOpenGLPolyDataMapper
{
    public:
        static vtk441PolyDataMapper *New();
        virtual void RenderPiece(vtkRenderer *ren, vtkActor *act)
        {
            float ambient[3] = {1, 1, 1};
            glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, ambient);
            glBegin(GL_TRIANGLES);
            glVertex3f(0, 0, 0);
            glVertex3f(0, 1, 0);
            glVertex3f(1, 0, 0);
            glVertex3f(1, 1, 0);
            glEnd();
        }
};
First OpenGL programs

class vtk441PolyDataMapper : public vtkOpenGLPolyDataMapper
{
    public:
        static vtk441PolyDataMapper *New();
        virtual void RenderPiece(vtkRenderer *ren, vtkActor *act)
        {
            glEnable(GL_COLOR_MATERIAL);
            float ambient[3] = { 1, 1, 1 };     
            glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, ambient);
            glBegin(GL_TRIANGLES);
            glColor3ub(0, 0, 255);
            glVertex3f(0,0,0);
            glVertex3f(0,1,0);
            glVertex3f(1,1,0);
            glEnd();
        }
};
glEnable/glDisable: important functions

Both glEnable and glDisable take a single argument, cap, which can assume one of the following values:

- **GL_BLEND**
  - If enabled, blend the computed fragment color values with the values in the color buffers. See glEnableFunc.
- **GL_CULL_FACE**
  - If enabled, cull polygons based on their winding in window coordinates. See glCullFace.
- **GL_DEPTH_TEST**
  - If enabled, do depth comparisons and update the depth buffer. Note that even if the depth buffer exists and the depth mask is non-zero, the depth buffer is not updated if the depth test is disabled. See glDepthFunc and glDepthRange.
- **GL_DITHER**
  - If enabled, dither color components or indices before they are written to the color buffer.
- **GL_POLYGON_OFFSET_FILL**
  - If enabled, an offset is added to depth values of a polygon’s fragments produced by rasterization. See glPolygonOffset.
- **GL_SAMPLE_ALPHA_TO_COVERAGE**
  - If enabled, compute a temporary coverage value where each bit is determined by the alpha value at the corresponding sample location. The temporary coverage value is then ANDed with the fragment coverage value.
- **GL_SAMPLE_COVERAGE**
  - If enabled, the fragment’s coverage is ANDed with the temporary coverage value. If GL_SAMPLE_COVERAGE_INVERT is set to GL_TRUE, invert the coverage value. See glSampleCoverage.
- **GL_SCISSOR_TEST**
  - If enabled, discard fragments that are outside the scissor rectangle. See glScissor.
- **GL_STENCIL_TEST**
  - If enabled, do stencil testing and update the stencil buffer. See glStencilFunc and glStencilOp.
class vtk441PolyDataMapper : public vtkOpenGLPolyDataMapper
{
    public:
        static vtk441PolyDataMapper *New();
        virtual void RenderPiece(vtkRenderer *ren, vtkActor *act)
        {
            glEnable(GL_COLOR_MATERIAL);
            float ambient[3] = {1, 1, 1};
            glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, ambient);
            glBegin(GL_TRIANGLES);
            glColor3ub(0, 0, 255);
            glVertex3f(0, 0, 0);
            glColor3ub(0, 255, 0);
            glVertex3f(0, 1, 0);
            glColor3ub(255, 0, 0);
            glVertex3f(1, 1, 0);
            glEnd();
        }
};
Visualization use case

Why is there purple in this picture?
First OpenGL programs

class vtk441PolyDataMapper : public vtkOpenGLPolyDataMapper
{
  public:
    static vtk441PolyDataMapper *New();
    virtual void RenderPiece(vtkRenderer *ren, vtkActor *act)
    {
      glEnable(GL_COLOR_MATERIAL);
      float ambient[3] = { 1, 1, 1 };
      glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, ambient);
      glBegin(GL_TRIANGLES);
      glColor3ub(0, 0, 255);
      glVertex3f(0,0,0);
      glColor3ub(0, 255, 0);
      glVertex3f(0,1,0);
      glColor3ub(255, 0, 0);
      glVertex3f(1,1,0);
      glEnd();
    }
};
Textures: a better way to specify a color map

There is no purple when we use textures
1D textures: basic idea

- Store color map on GPU as a texture

- Old color interpolation of fragment on a scanline:
  - For (int \( j = 0 \); \( j < 3 \); \( j++ \))
    - \( \text{RGB}[j] = \text{leftRGB}[j] + \text{proportion} \times (\text{rightRGB}[j]-\text{leftRGB}[j]) \)

- New color interpolation of fragment on a scanline:
  - \( \text{textureVal} = \text{leftTextureVal} \)
    - \(+ \text{proportion} \times (\text{rightTextureVal}-\text{leftTextureVal}) \)
  - \( \text{RGB} \leftarrow \text{textureLookup}[\text{textureVal}] \)
Example

- Triangle with vertices with scalar values 2.9, 3.3, and 3.1.
- $T$ for 2.9 = $(2.9-3.0)/(3.25-3) = -0.4$
- $T$ for 3.1 = $(3.1-3.0)/(3.25-3) = 0.4$
- $T$ for 3.3 = $(3.3-3.0)/(3.25-3) = 1.2$
- Fragment colors come from interpolating texture coordinates and applying texture
class vtk441PolyDataMapper : public vtkOpenGLPolyDataMapper {
  public:
    static vtk441PolyDataMapper *New();

    virtual void RenderPiece(vtkRenderer *, vtkActor *) {
      GLubyte Texture3[9] = {
        0, 0, 255,  // blue
        255, 255, 255,  // white
        255, 0, 0,  // red
      };
      glTexImage1D(GL_TEXTURE_1D, 0, GL_RGB, 3, 0, GL_RGB,
                   GL_UNSIGNED_BYTE, Texture3);
      glEnable(GL_COLOR_MATERIAL);
      glTexParameterf(GL_TEXTURE_1D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
      glTexParameterf(GL_TEXTURE_1D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
      glEnable(GL_TEXTURE_1D);
      float ambient[] = {1, 1, 1};
      glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, ambient);
      glBegin(GL_TRIANGLES);
      glTexCoord1f(0);
      glVertex3f(0, 0, 0);
      glTexCoord1f(0.0);
      glVertex3f(0, 1, 0);
      glTexCoord1f(1.0);
      glVertex3f(1, 1, 0);
      glEnd();
    }
};
class vtk441PolyDataMapper : public vtkOpenGLPolyDataMapper
{
    public:
        static vtk441PolyDataMapper *New();

    virtual void RenderPiece(vtkRenderer *ren, vtkActor *act)
    {
        GLubyte Texture3[9] = {
            0, 0, 255, // blue
            255, 255, 255, // white
            255, 0, 0, // red
        };
        glTexImage1D(GL_TEXTURE_1D, 0, GL_RGB, 3, 0, GL_RGB,
                     GL_UNSIGNED_BYTE, Texture3);
        glEnable(GL_COLOR_MATERIAL);
        glTexParameterf(GL_TEXTURE_1D, GL_TEXTURE_WRAP_S, GL_CLAMP);
        glTexParameterf(GL_TEXTURE_1D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

        glEnable(GL_TEXTURE_1D);
        float ambient[3] = { 1, 1, 1 };
        glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, ambient);
        glBegin(GL_TRIANGLES);
        glTexCoord1f(0);
        glVertex3f(0,0,0);
        glTexCoord1f(0.0);
        glVertex3f(0,1,0);
        glTexCoord1f(1.);
        glVertex3f(1,1,0);
        glEnd();
    }
};
Texture wrapping options

- Sets the wrap parameter for texture coordinate s to GL_CLAMP, GL_REPEAT, GL_CLAMP_TO_BORDER_EXT, or GL_CLAMP_TO_EDGE_EXT.

- GL_CLAMP causes s coordinates to be clamped to the range [0, 1] and is useful for preventing wrapping artifacts when mapping a single image onto an object.

- GL_REPEAT causes the integer part of the s coordinate to be ignored; the GL uses only the fractional part, thereby creating a repeating pattern.
Texture wrapping options

- **GL_CLAMP_TO_BORDER_EXT** causes s coordinates to be clamped to a range $1/2$ texel outside $[0, 1]$; this prevents the "half border, half edge" color artifact.
- **GL_CLAMP_TO_EDGE_EXT** causes s coordinates to be clamped to a range $1/2$ texel inside $[0, 1]$; this prevents any border colors from showing up in the image.
- Border texture elements are accessed only if wrapping is set to **GL_CLAMP** or **GL_CLAMP_TO_BORDER_EXT**. Initially, **GL_TEXTURE_WRAP_S** is set to **GL_REPEAT**.
Textures with GL_REPEAT

class vtk441PolyDataMapper : public vtkOpenGLPolyDataMapper
{
  public:
    static vtk441PolyDataMapper *New();

    virtual void RenderPiece(vtkRenderer *ren, vtkActor *act)
    {
      GLubyte Texture3[9] = {
        0, 0, 255, // blue
        255, 255, 255, // white
        255, 0, 0, // red
      };
      glTexImage1D(GL_TEXTURE_1D, 0, GL_RGB, 3, 0, GL_RGB,
                   GL_UNSIGNED_BYTE, Texture3);
      glEnable(GL_COLOR_MATERIAL);
      glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_WRAP_S, GL_REPEAT);
      glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
      glEnable(GL_TEXTURE_1D);
      float ambient[3] = { 1, 1, 1 };
      glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, ambient);
      glBegin(GL_TRIANGLES);
      glVertex3f(-2);
      glVertex3f(0,0,0);
      glVertex3f(0,0,0);
      glVertex3f(0,1,0);
      glVertex3f(0,1,0);
      glVertex3f(4.);
      glVertex3f(1,1,0);
      glEnd();
    }
};
1D, 2D, 3D textures

- 2D textures most common
- 1D textures: color maps (e.g., what we just did)
- 3D textures: “volume rendering”
  - Use combination of opacity and color (i.e., RGBA)
2D Textures

- Pre-rendered images painted onto geometry
- `glTexImage1D` → `glTexImage2D`
- `GL_TEXTURE_WRAP_S` → `GL_TEXTURE_WRAP_S` + `GL_TEXTURE_WRAP_T`
- `glTexCoord1f` → `glTexCoord2f`
2D Texture Program

```cpp
class vtk441PolyDataMapper : public vtkOpenGLPolyDataMapper
{
public:
    static vtk441PolyDataMapper *New();

    virtual void RenderPiece(vtkRenderer *ren, vtkActor *act)
    {
        vtkJPEGReader *rdr = vtkJPEGReader::New();
        rdr->SetFileName("HankChilds_345.jpg");
        rdr->Update();
        vtkImageData *img = rdr->GetOutput();
        int dims[3];
        img->GetDimensions(dims);
        unsigned char *buffer = (unsigned char *) img->GetScalarPointer(0, 0, 0);
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, dims[0], dims[1], 0, GL_RGB, GL_UNSIGNED_BYTE, buffer);
        glEnable(GL_COLOR_MATERIAL);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
        glDisable(GL_TEXTURE_2D);
        float ambient[3] = {1, 1, 1};
        glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, ambient);
        glBegin(GL_TRIANGLES);
        glTexCoord2f(0, 0);
        glVertex3f(0, 0, 0);
        glVertex3f(0, 1, 0);
        glVertex3f(1, 0, 0);
        glEnd();
    }
};
```
Minifying: texture bigger than triangle.

- How to map multiple texture elements onto a pixel?
  - GL_NEAREST: pick closest texture
  - GL_LINEAR: average neighboring textures

Magnifying (GL_TEXTURE_MAG_FILTER): triangle bigger than texture

- How to map single texture element onto multiple pixels?
  - GL_NEAREST: no interpolation
  - GL_LINEAR: interpolate with neighboring textures
GL_TEXTURE_MAG_FILTER with NEAREST and LINEAR

class vtk441PolyDataMapper : public vtkOpenGLPolyDataMapper
{
  public:
    static vtk441PolyDataMapper *New();

    virtual void RenderPiece(vtkRenderer *ren, vtkActor *act)
    {
      GLubyte Texture3[9] = {
        0, 0, 255,    // blue
        255, 255, 255, // white
        255, 0, 0,    // red
      };
      glTexImage1D(GL_TEXTURE_1D, 0, GL_RGB, 3, 0, GL_RGB, GL_UNSIGNED_BYTE, Texture3);
      glEnable(GL_COLOR_MATERIAL);
      glTexParameterf(GL_TEXTURE_1D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
      ______
      glTexParameterf(GL_TEXTURE_1D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
      --- OR ---
      glTexParameterf(GL_TEXTURE_1D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
      ______
      glTexParameterf(GL_TEXTURE_1D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

      glEnable(GL_TEXTURE_1D);
      float ambient[3] = { 1, 1, 1 };
      glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, ambient);
      glBegin(GL_TRIANGLES);
      glTexCoord1f(0.);
      glVertex3f(0,0,0);
      glVertex3f(0,0,0);
      glVertex3f(0,1,0);
      glVertex3f(1,0);
      glVertex3f(1,1,0);
      glEnd();
    }
};
2D Texture Program

Texture is not 1:1, should probably scale geometry.

This is a terrible program ... why?
glBindTexture: tell the GPU about the texture once and re-use it!
Mipmaps: pre-calculated, optimized collections of images that accompany a main texture, intended to increase rendering speed and reduce aliasing artifacts.

Widely used in 3D computer games, flight simulators and other 3D imaging systems.

In use, it is called “mipmapping.”

The letters "MIP" in the name are an acronym of the Latin phrase multum in parvo, meaning "much in little".
Mipmaps