L11ComputerGraphics

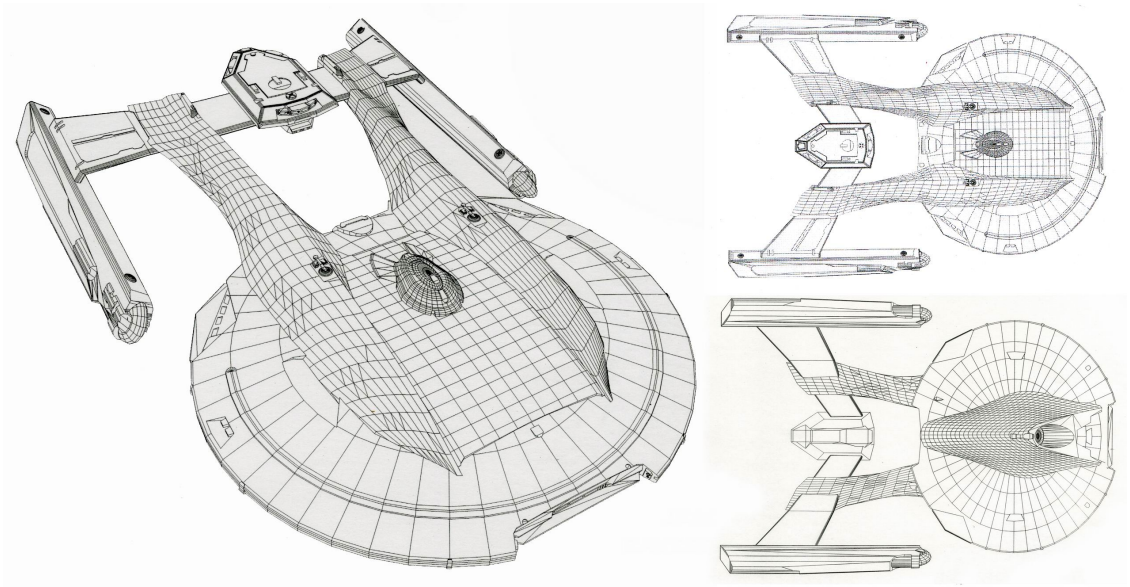
March 15, 2015

1 Computer Graphics

Today we'll study the mathematics used to create and manipulate computer graphics.
The computer graphics seen in movies and videogames works in three stages:

1. A 3D model of the scene objects is created;
2. The model is converted into (many small) polygons in 3D that approximate the surfaces of the model; and
3. The polygons are transformed via a linear transformation to yield a 2D representation that can be shown on a flat screen.

There is interesting mathematics in each stage, but the transformations that take place in the third stage are **linear**, and that's what we'll study today.

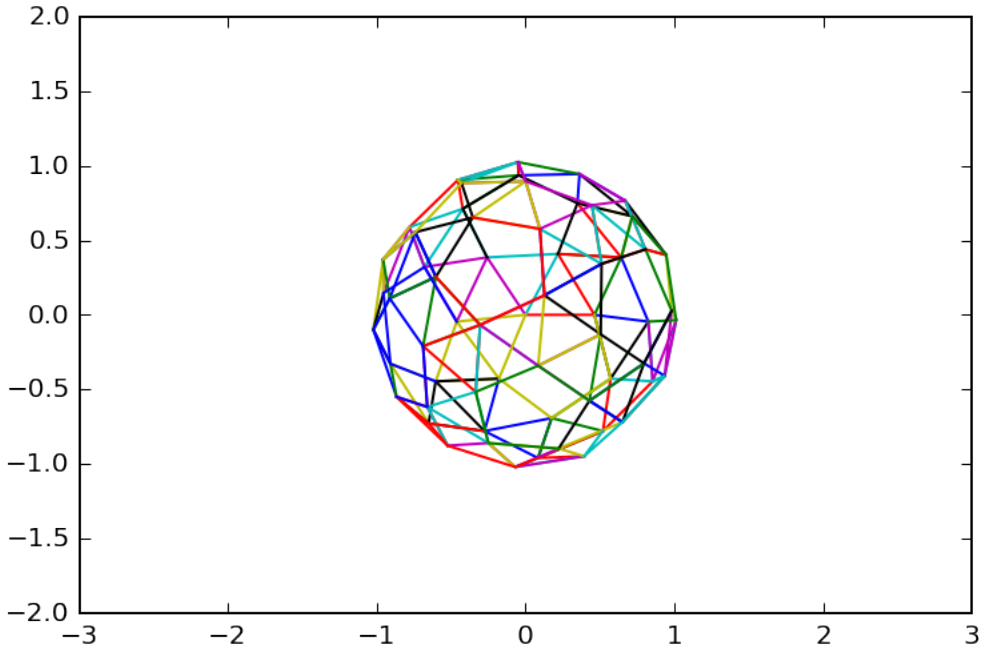


Initially, object models may be expressed in terms of smooth functions like polynomials.

However the first step is to convert those smooth functions into a set of discrete pieces – coordinates and line segments. All subsequent processing is done in terms of the discrete coordinates that approximate the shape of the original model.

The reason for this conversion is that most transformations needed in graphics are *linear*. Expressing the scene in terms of coordinates is equivalent to expressing it in terms of vectors, eg, in $\mathbb{R}^3$. And linear transformations on vectors are always matrix multiplications, so implementation is simple and uniform.

The resulting representation consists of lists of 3D coordinates called *faces.* Each face is a polygon.
The lines drawn between coordinates are implied by the way that coordinates are grouped into faces.
Example.
Here is a view of a ball-like object. It is centered at the origin.
The ball is represented in terms of 3D coordinates, but we are only plotting the x and y coordinates here.
Colors correspond to faces.

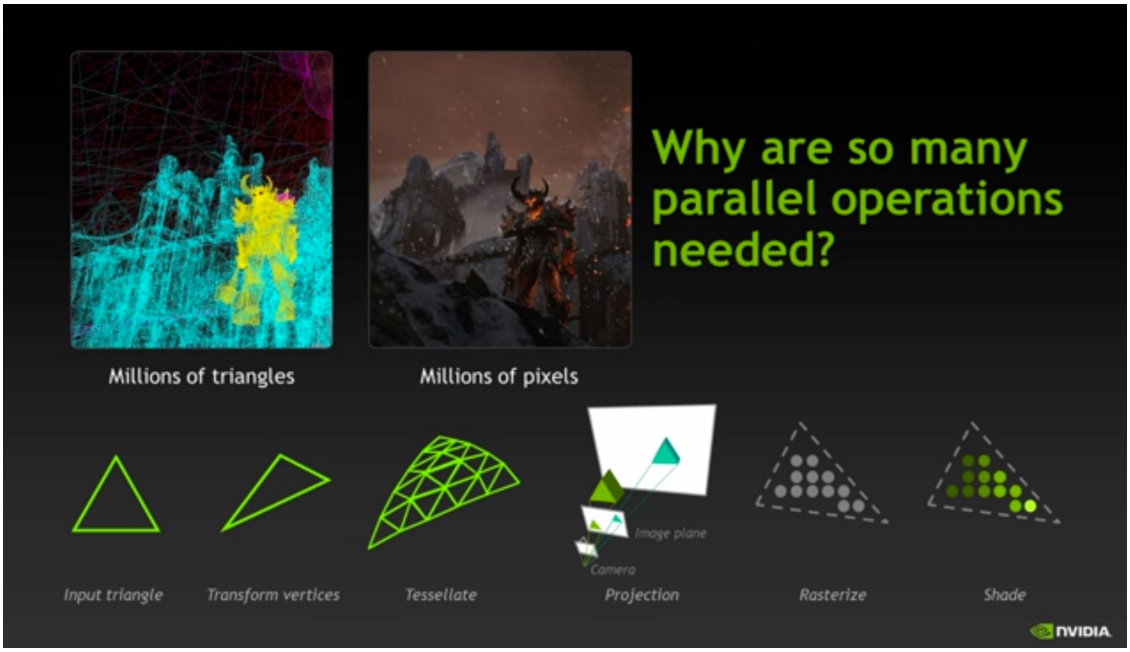


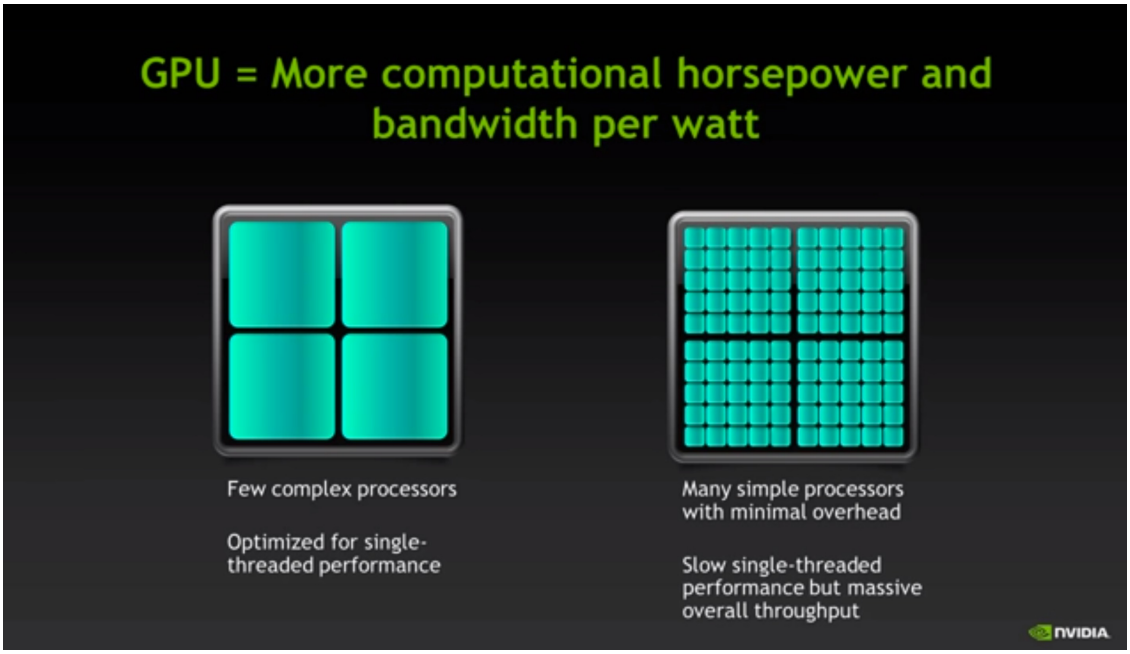
Imagine that we want to circle the camera around the ball. This is equivalent to rotating the ball around the y axis.

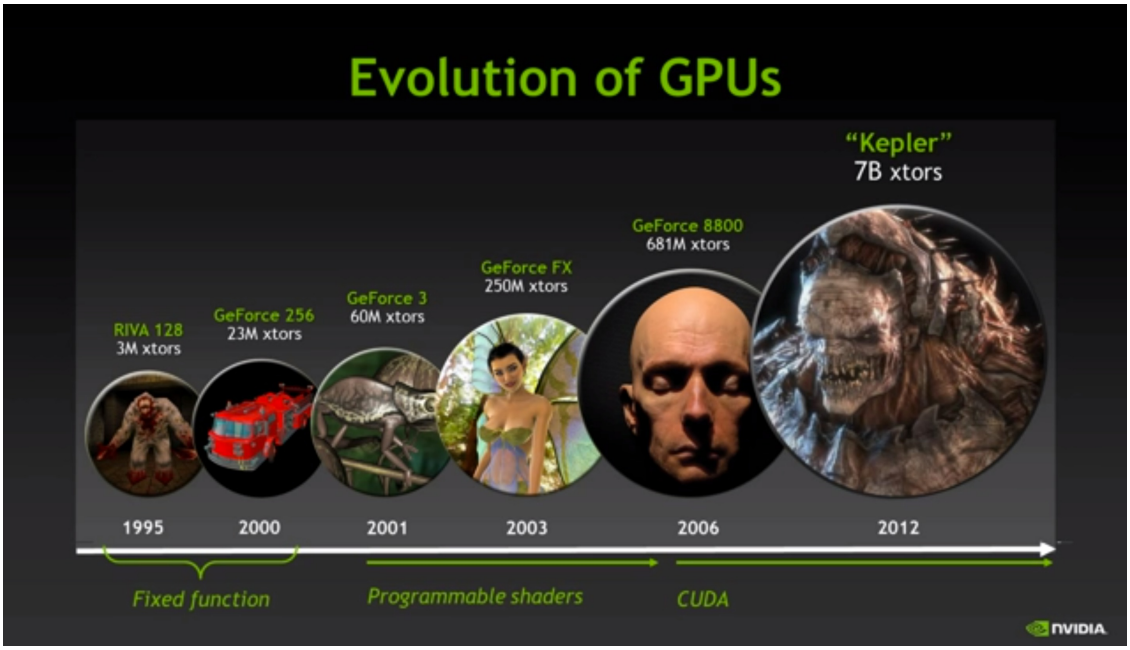
```
In [15]: # set up view
         import matplotlib.animation as animation
         fig = plt.figure()
         ax = plt.axes(xlim=(-5,5),ylim=(-5,5))
         plt.plot(-2,-2,'')
         plt.plot(2,2,'')
         plt.axis('equal')
         ballLines = []
         for b in ball:
             ballLines += ax.plot([],[])
         def animate(i):
             angle = 2.0 * np.pi * (i/100.0)
             rotationMatrix = np.array([[np.cos(angle),0,-np.sin(angle)],[0,1,0],[np.sin(angle),0,np.cos
             for b,l in zip(ball,ballLines):
                 rb = rotationMatrix.dot(b)
                 l.set_data(rb[0],rb[1])
             fig.canvas.draw()
         anim = animation.FuncAnimation(fig,animate,frames=100,interval=100,repeat=False,blit=False)
         # this function requires ffmpeg to be installed on your system
         display_animation(anim)
```

1.1 Fast Computation of Linear Systems and Graphics Processing Units (GPUs)

In [7]: `# Note: Another good source is http://pixeljetstream.blogspot.de/2015/02/life-of-triangle-nvidi`



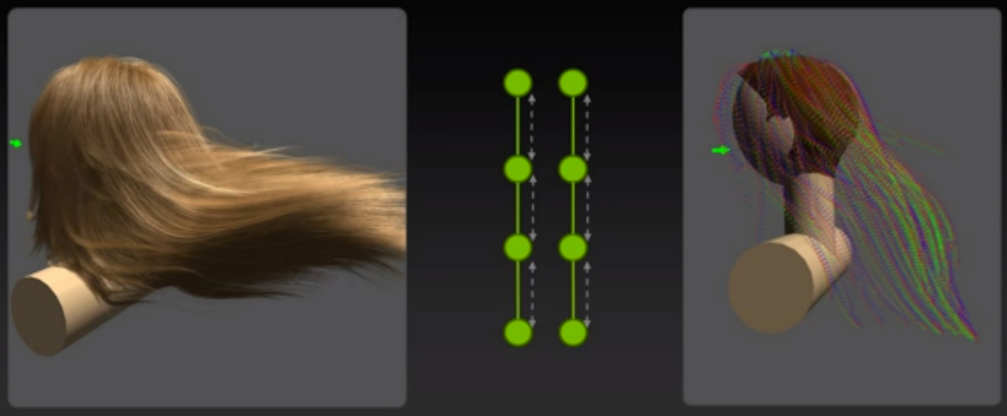




Examples of Other Linear Systems Used in Computer Graphics

PARTICLE SIMULATION — GAMING

Hair simulation

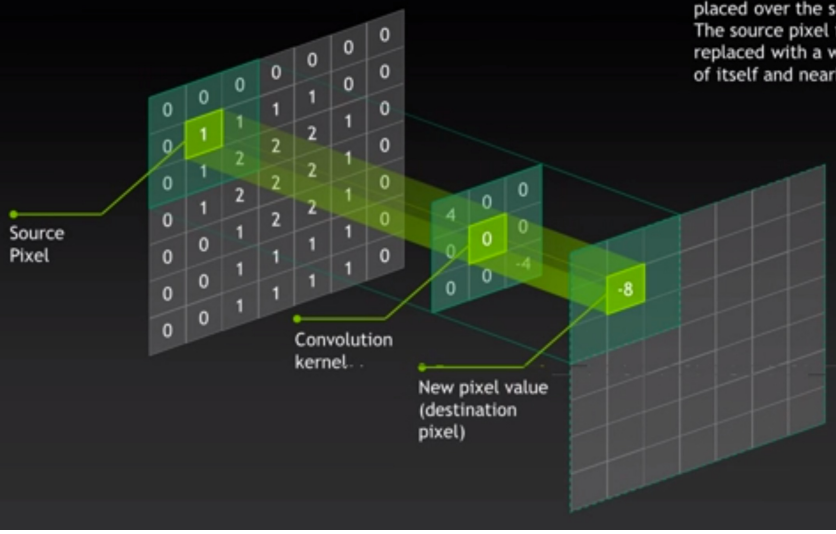


NVIDIA Hair Demo

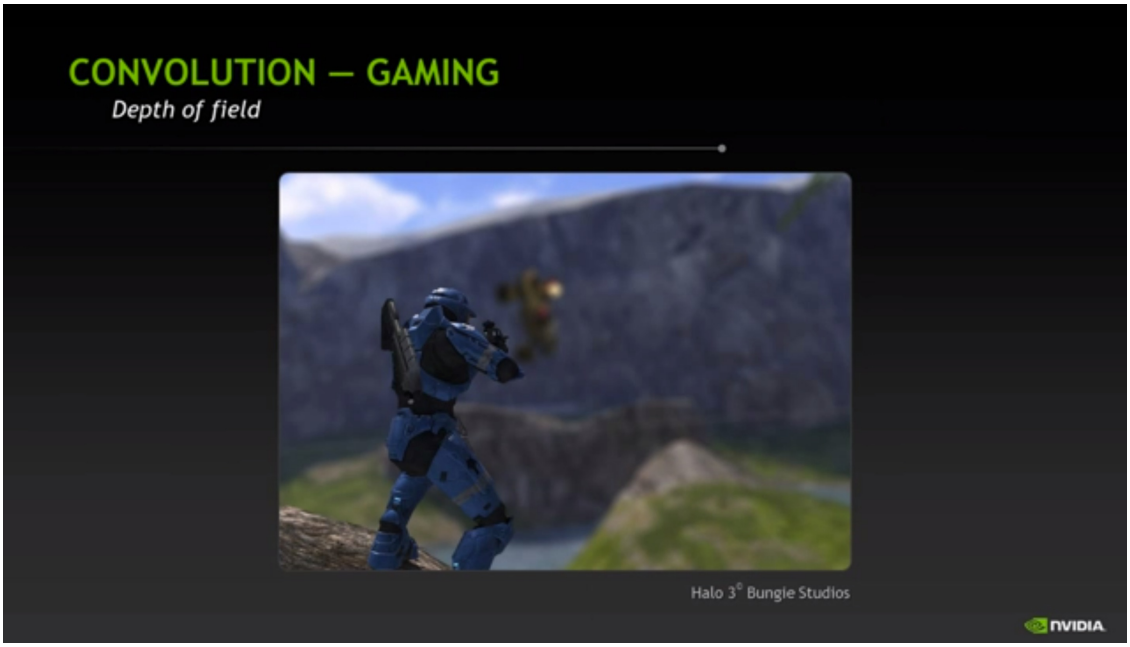
NVIDIA

2| CONVOLUTION

Center element of the kernel is placed over the source pixel. The source pixel is then replaced with a weighted sum of itself and nearby pixels.



NVIDIA


CONVOLUTION — GAMING
Depth of field
Halo 3© Bungie Studios
NVIDIA

In [14]: