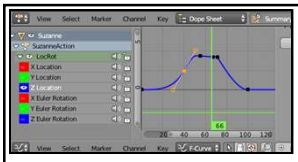
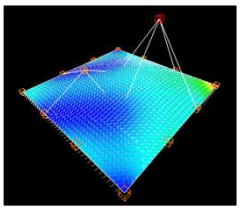
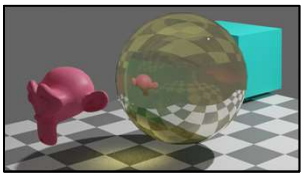


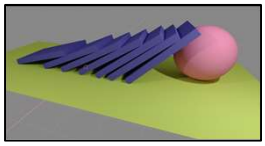
SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

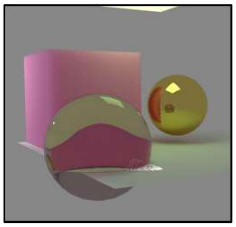
A Whirlwind Introduction to Computer Graphics

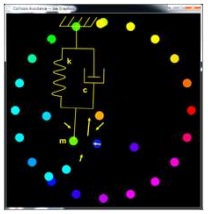


Mike Bailey
mjb@cs.oregonstate.edu









http://cs.oregonstate.edu/~mjb/whirlwind

SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

Mike Bailey
Oregon State University

mjb@cs.oregonstate.edu

http://cs.oregonstate.edu/~mjb/whirlwind

Course Goals

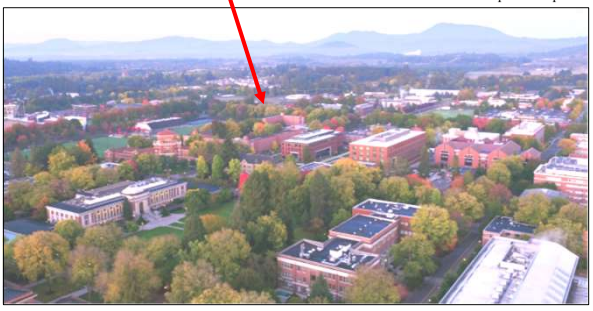
- **Provide a background for the amazing things you will hear about in the other SIGGRAPH 2023 venues**
- **Create an understanding of common computer graphics vocabulary**
- **Help you understand the significance of the images and animations that you will see**
- **Provide references for further study**

http://cs.oregonstate.edu/~mjb/whirlwind

Mike Bailey

- Professor of Computer Science, Oregon State University
- Has been in computer graphics for over 30 years
- Has had over 11,000 students in his university classes
- Has taught over 100 conference and workshop short courses
- mjb@cs.oregonstate.edu

Oregon State University
Computer Graphics

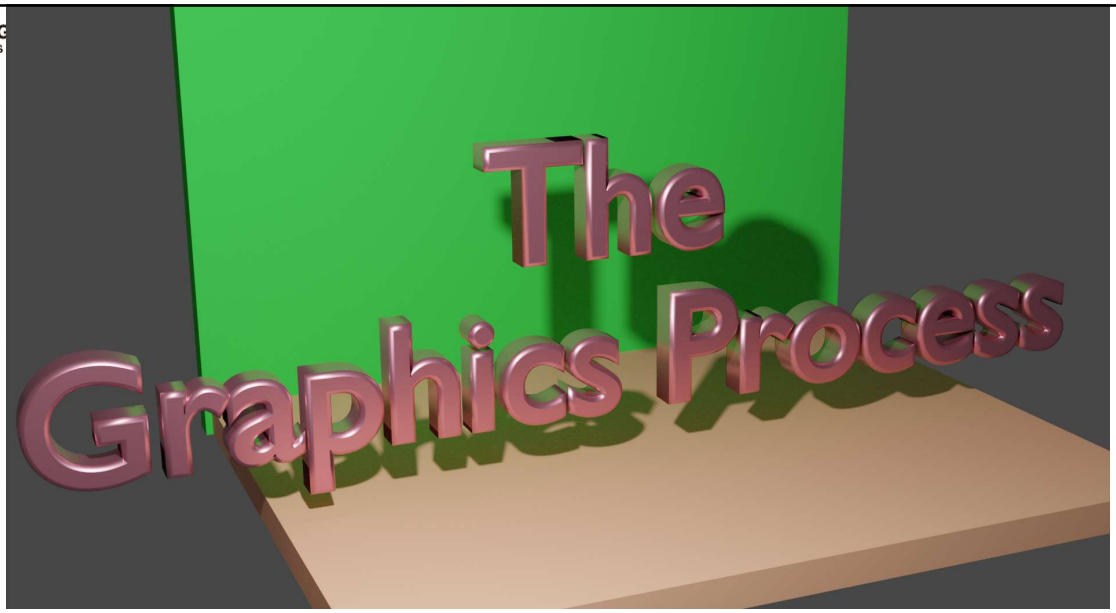


http://cs.oregonstate.edu/~mjb/whirlwind

Sections

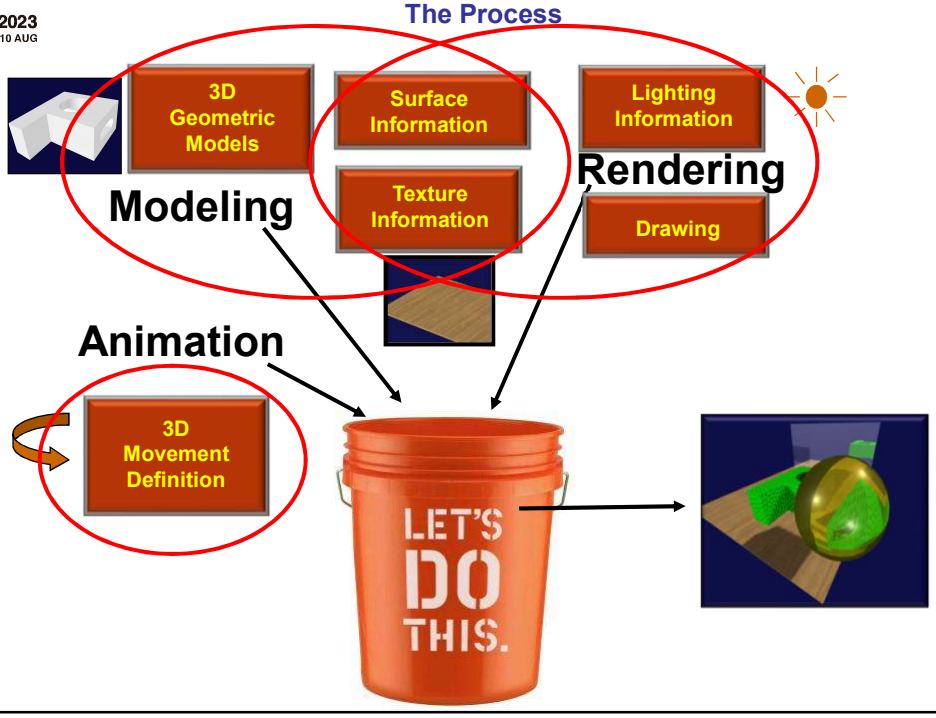
1. How the computer graphics pieces fit together
2. Modeling
3. Rendering
4. Animation
5. Finding More Information

http://cs.oregonstate.edu/~mjb/whirlwind

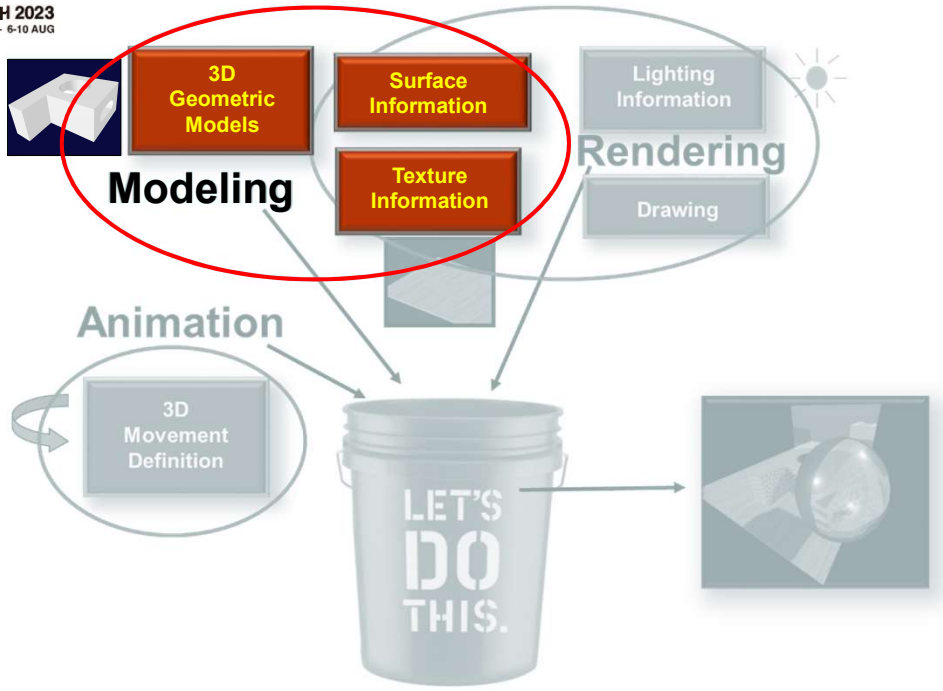


What are all the pieces that go into making the graphics you will be see?
What does it take to make them?

The Process



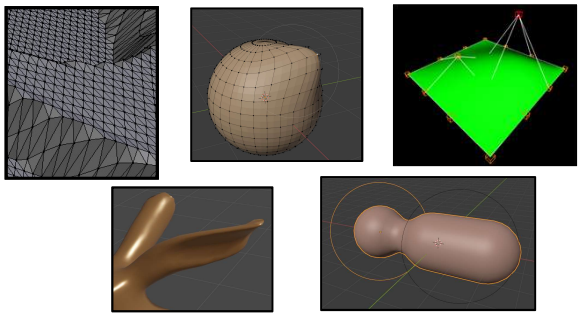
Creating 3D Geometry



What do we mean by "Modeling"?

In computer graphics applications, how we model geometry depends on what we would like to use the geometry for:

- Looking at its appearance
- Interacting with its shape?
- How does it interact with its environment?
- What is its surface area and volume?
- Will it be able to be 3D-printed?
- Etc.



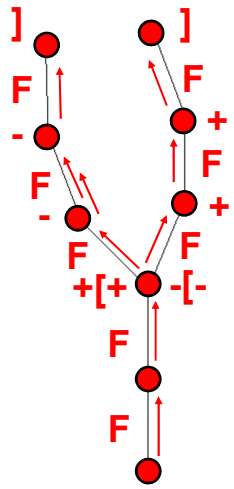
Want to experiment with some free modeling programs?
Want some notes on how to get started?
http://cs.oregonstate.edu/~mjb/blender
http://cs.oregonstate.edu/~mjb/sketchup
http://cs.oregonstate.edu/~mjb/tinkercad

L-Systems as a Special Way to Model 3D Line Geometry

Introduced and developed in 1968 by Aristid Lindenmayer, L-systems are a way to apply grammar rules for generating fractal (self-similar) geometric shapes. For example, take the string:

"FF+[+F-F-F]-[-F+F+F]"

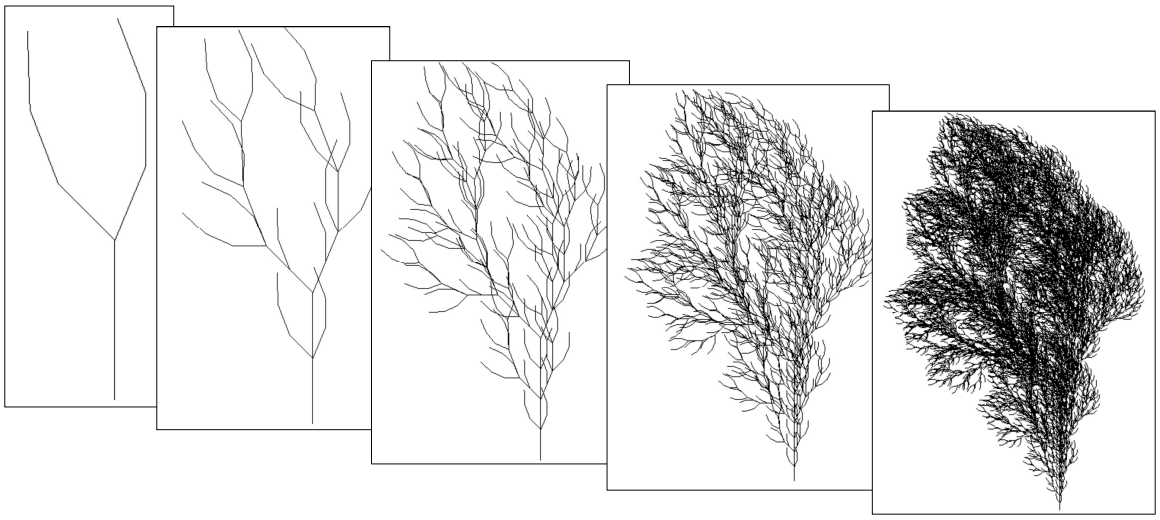
F	move forward one step
+	turn right
-	turn left
[	push state
]	pop state



L-Systems as a Special Way to Model 3D Line Geometry

But the *real* fun comes when you call that string recursively. For every **F**, replicate that string but with smaller geometry:

"F → FF+[+F-F-F]-[-F+F+F]"

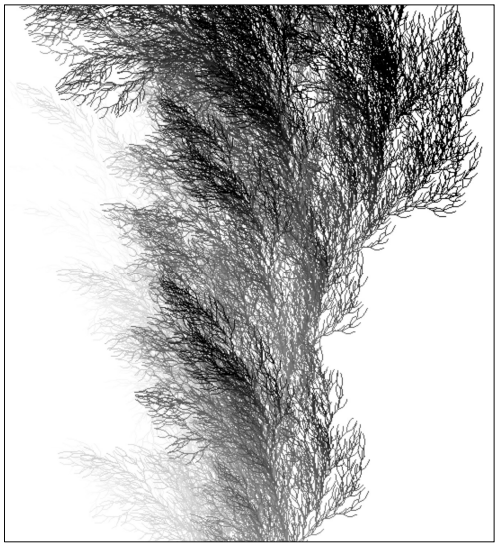


L-Systems as a Special Way to Model 3D Line Geometry

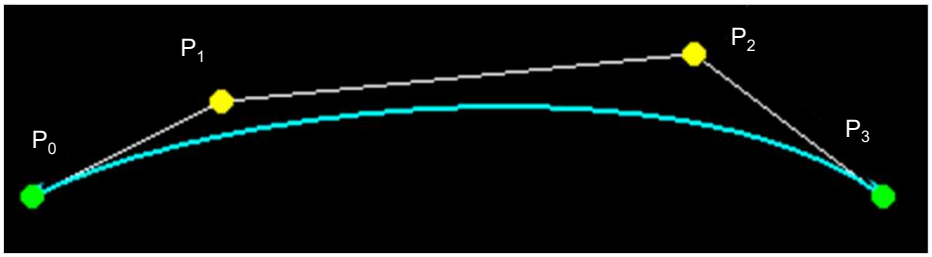
And, of course we can introduce more grammar to swing it into 3D

"F → FF+[+F-<F->F]-[-F+^F+vF]"

+	rotate + about Z
-	rotate - about Z
<	rotate + about Y
>	rotate – about Y
v	rotate + about X
^	rotate – about X



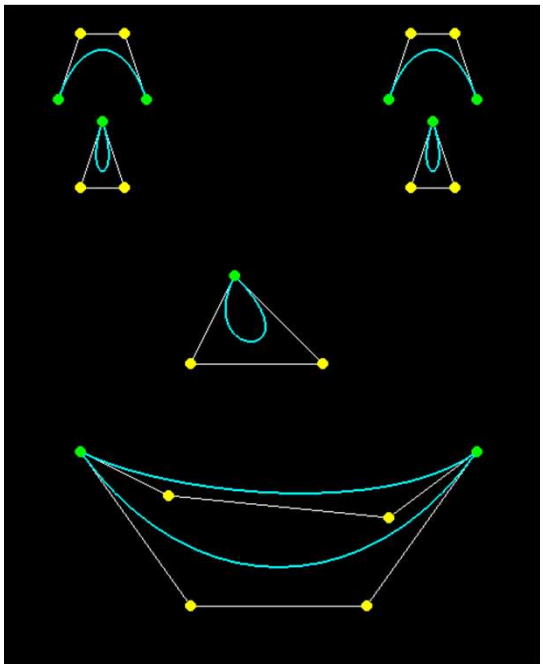
Another way to Model:
Curve Sculpting – Bézier Curve Sculpting



$$P(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3$$

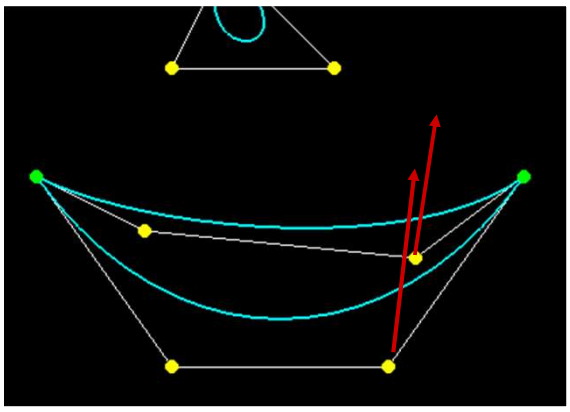
$$0. \le t \le 1.$$

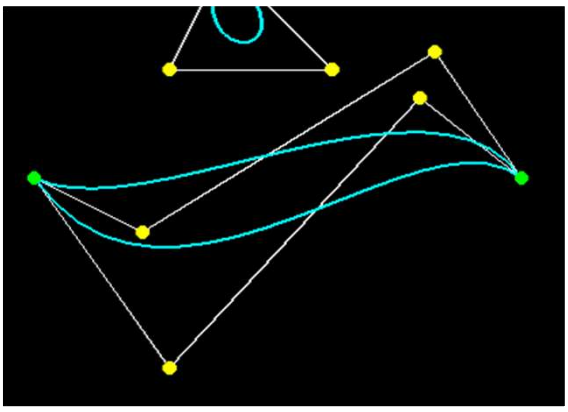
Curve Sculpting – Bézier Curve Sculpting Example



curves.mp4

Curve Sculpting – Bézier Curve Sculpting Example

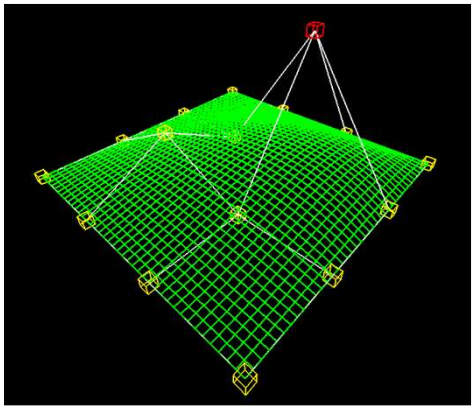




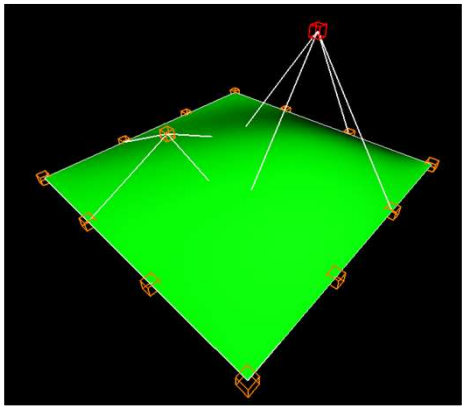
A *Small* Amount of Input Change Results in a *Large* Amount of Output Change

Another way to Model: Surface Sculpting

In general, these are referred to as ***Patches***. These, in particular, are Beziér patches. Non-uniform Rational B-spline Surfaces, or NURBS, are another popular type.



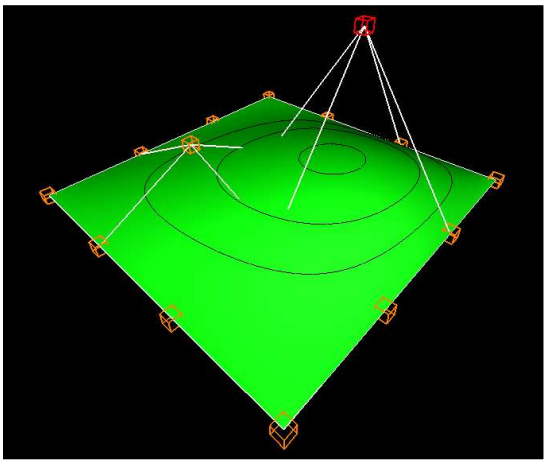
Wireframe



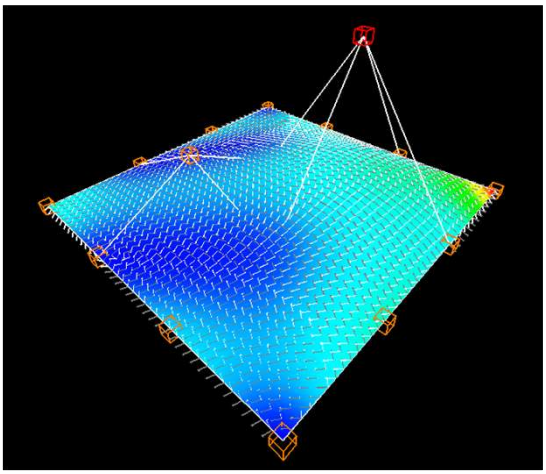
Surface

Like the curve sculpting, a *Small* Amount of Input Change Results in a *Large* Amount of Output Change

Surface Equations can also be used for Analysis



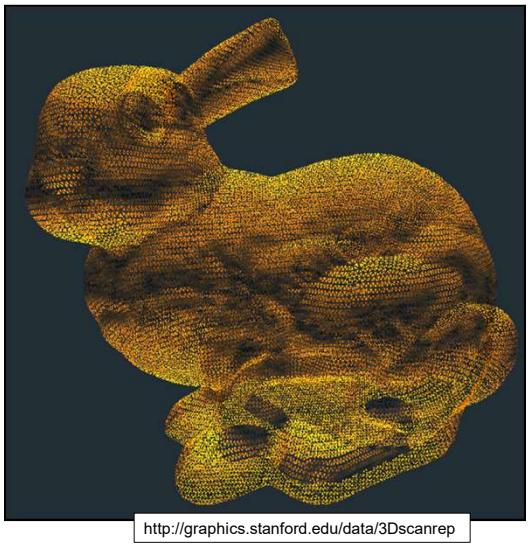
Showing Contour Lines



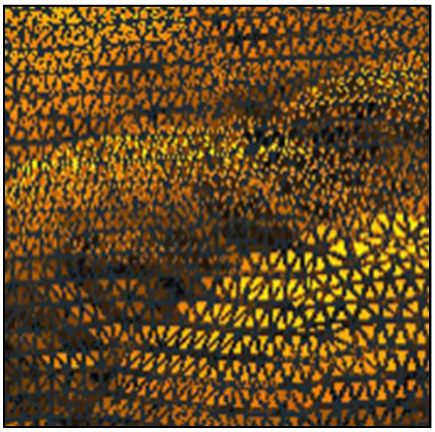
Showing Curvature

Explicitly Listing Geometry and Topology

Models defined this way can consist of thousands of vertices and faces – we need some way to describe them effectively

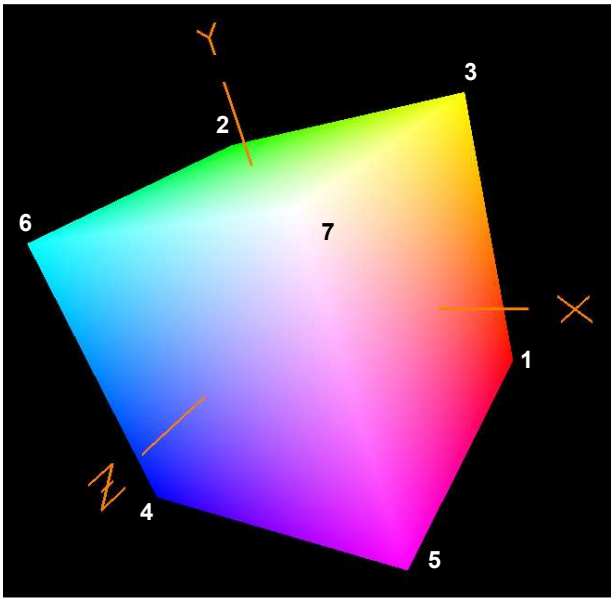


http://graphics.stanford.edu/data/3Dscanrep



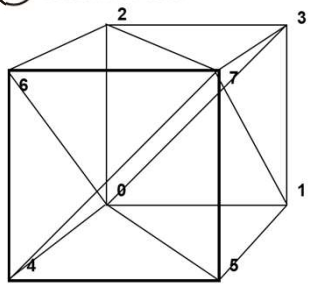
This is often called a **Mesh.**

Cube Mesh Example



SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

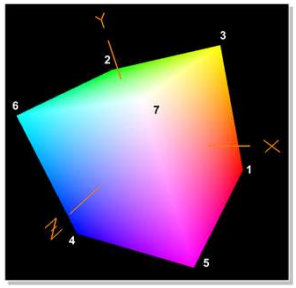
Explicitly Listing Geometry and Topology – Triangular Meshes




```
float CubeVertices[ ][3] =
{
        { -1., -1., -1. },
        {  1., -1., -1. },
        { -1.,  1., -1. },
        {  1.,  1., -1. },
        { -1., -1.,  1. },
        {  1., -1.,  1. },
        { -1.,  1.,  1. },
        {  1.,  1.,  1. }
};
```

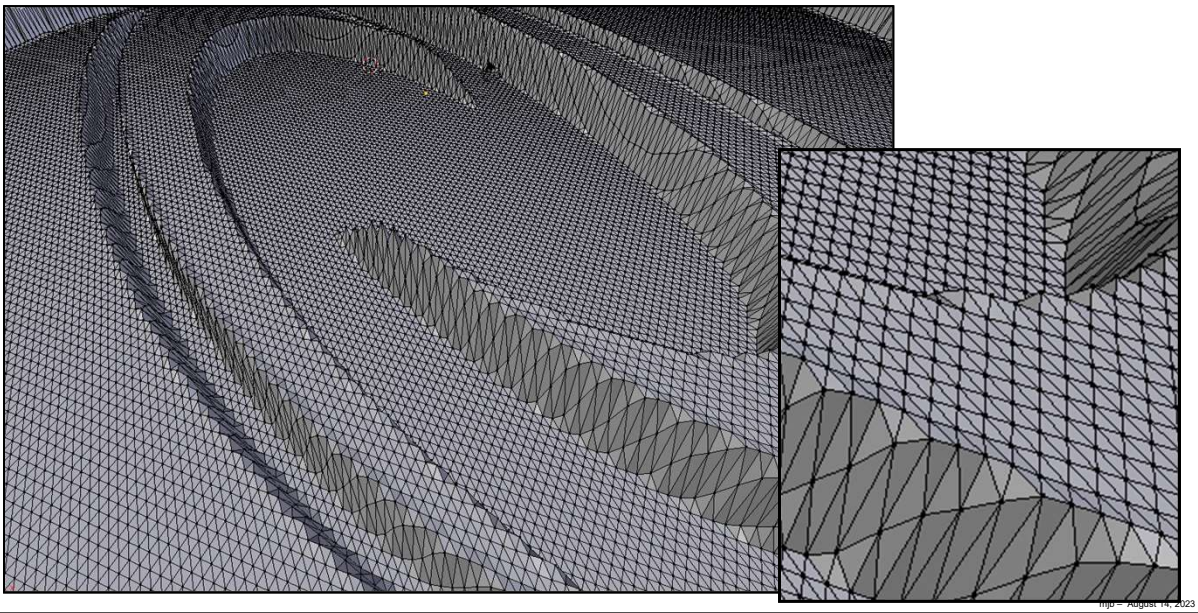
```
int CubeIndices[ ][3] =
{
        { 0, 2, 3 },
        { 0, 3, 1 },
        { 4, 5, 7 },
        { 4, 7, 6 },
        { 1, 3, 7 },
        { 1, 7, 5 },
        { 0, 4, 6 },
        { 0, 6, 2 },
        { 2, 6, 7 },
        { 2, 7, 3 },
        { 0, 1, 5 }
        { 0, 5, 4 }
};
```

```
float CubeColors[ ][3] =
{
        { 0., 0., 0. },
        { 1., 0., 0. },
        { 0., 1., 0. },
        { 1., 1., 0. },
        { 0., 0., 1. },
        { 1., 0., 1. },
        { 0., 1., 1. },
        { 1., 1., 1. },
};
```

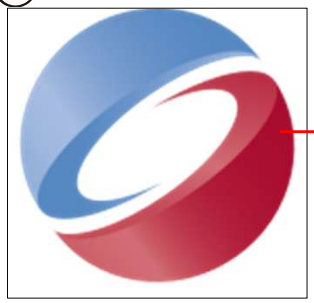


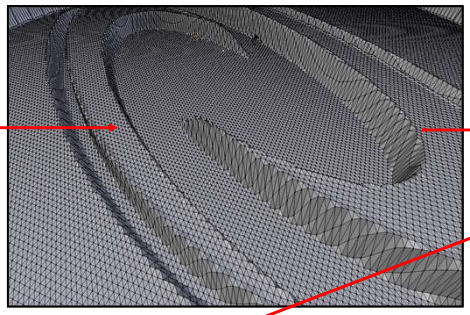
SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

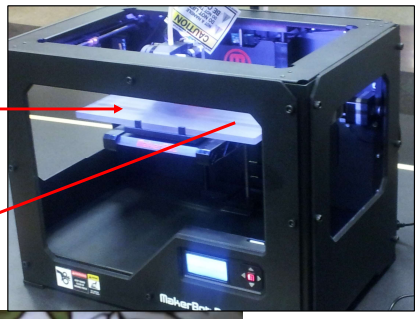
Triangular Meshes are Extra Important These Days Because 3D Printing Requires a Triangular Mesh Data Format

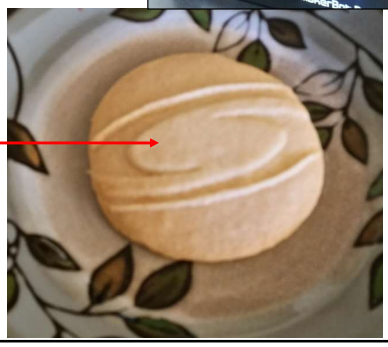


3D geometric modeling at its very best -- mmmm… :-)

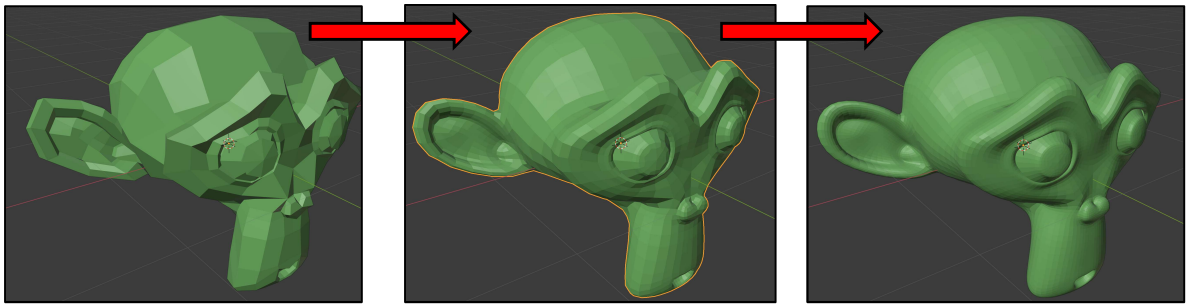




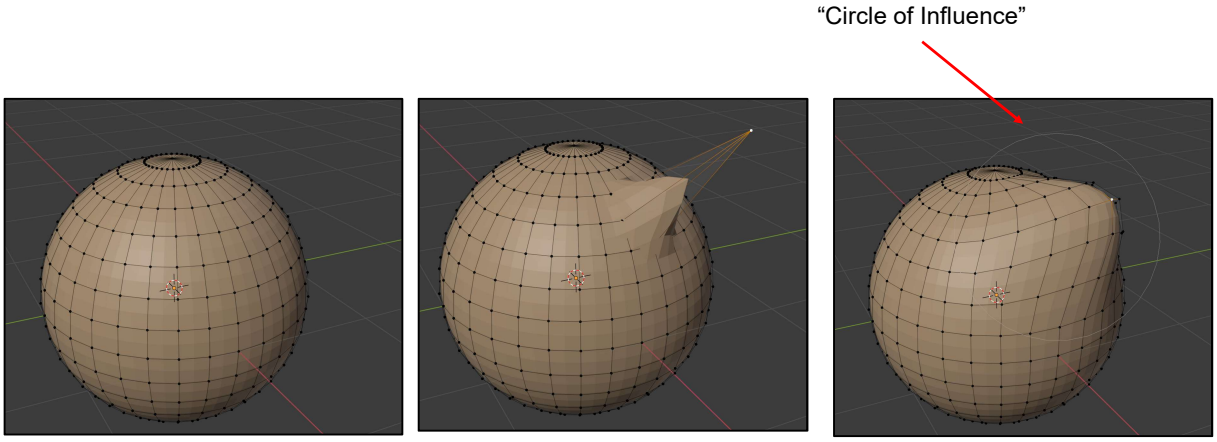




Meshes Can Be Smoothed



Meshes Can Be Edited

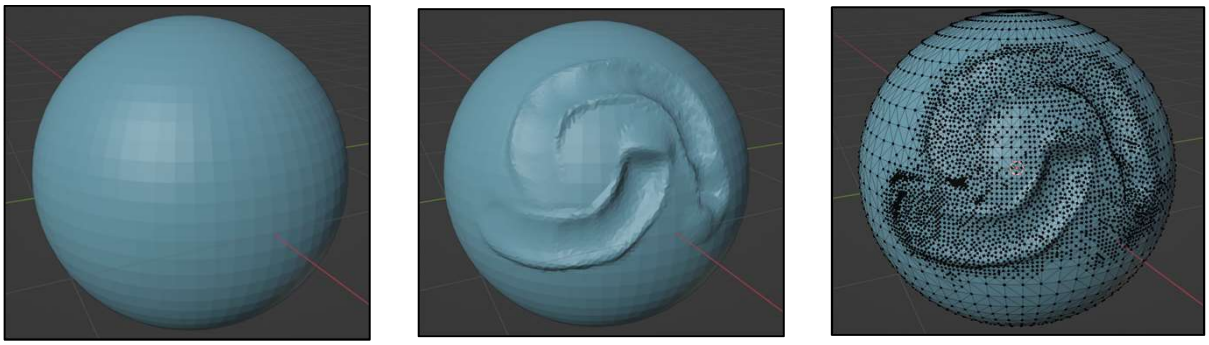


Original

Pulling on a single Vertex

Pulling on a Vertex with Proportional Editing Turned On

Meshes Can Be Sculpted

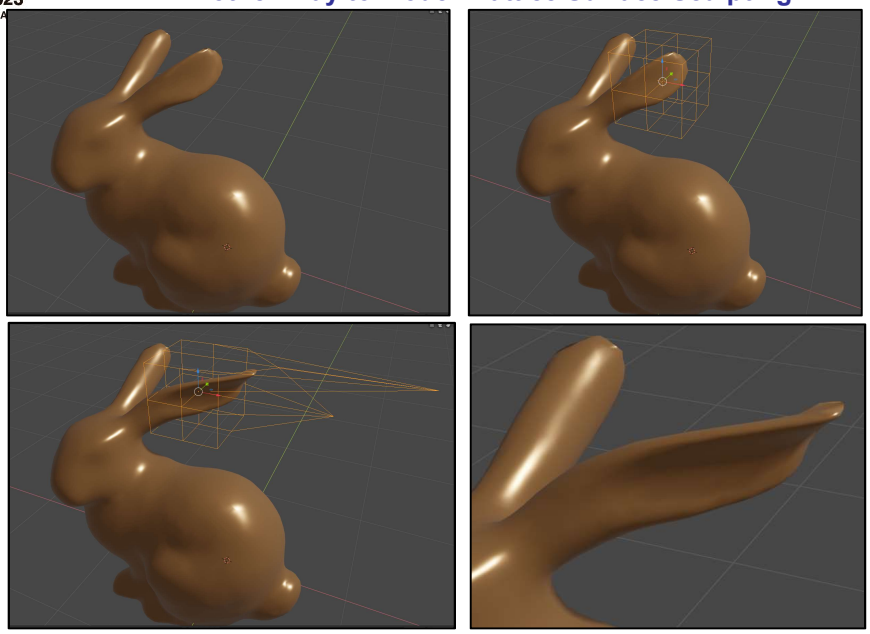


Original

“Clay Thumb” Sculpting

Sculpting Can Produce Extra Mesh Vertices

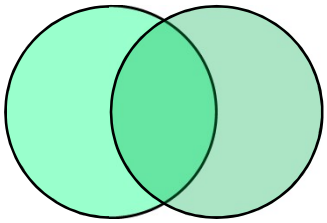
Another Way to Model: Lattice Surface Sculpting



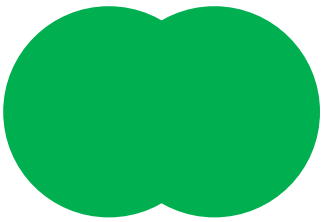
lattice.mp4

This is often called a **"Lattice"** or a **"Cage"**.

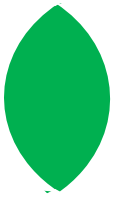
Another way to Model:
Remember Venn Diagrams (2D Boolean Operators)?



Two Overlapping Shapes



Union



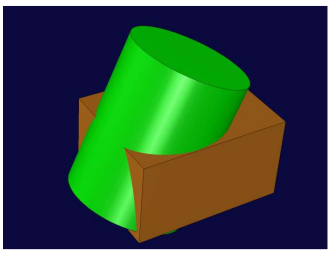
Intersection

Difference

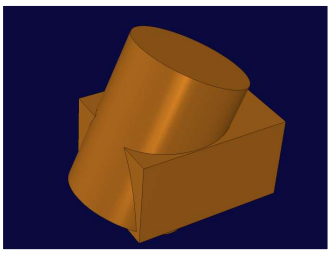
Blech! I thought I left Venn Diagrams behind in High School !

29

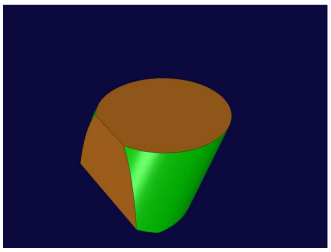
Solid Modeling Using 3D Boolean Operators



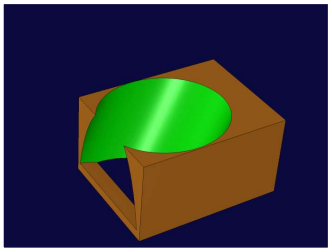
Two Overlapping Solids



Union



Intersection

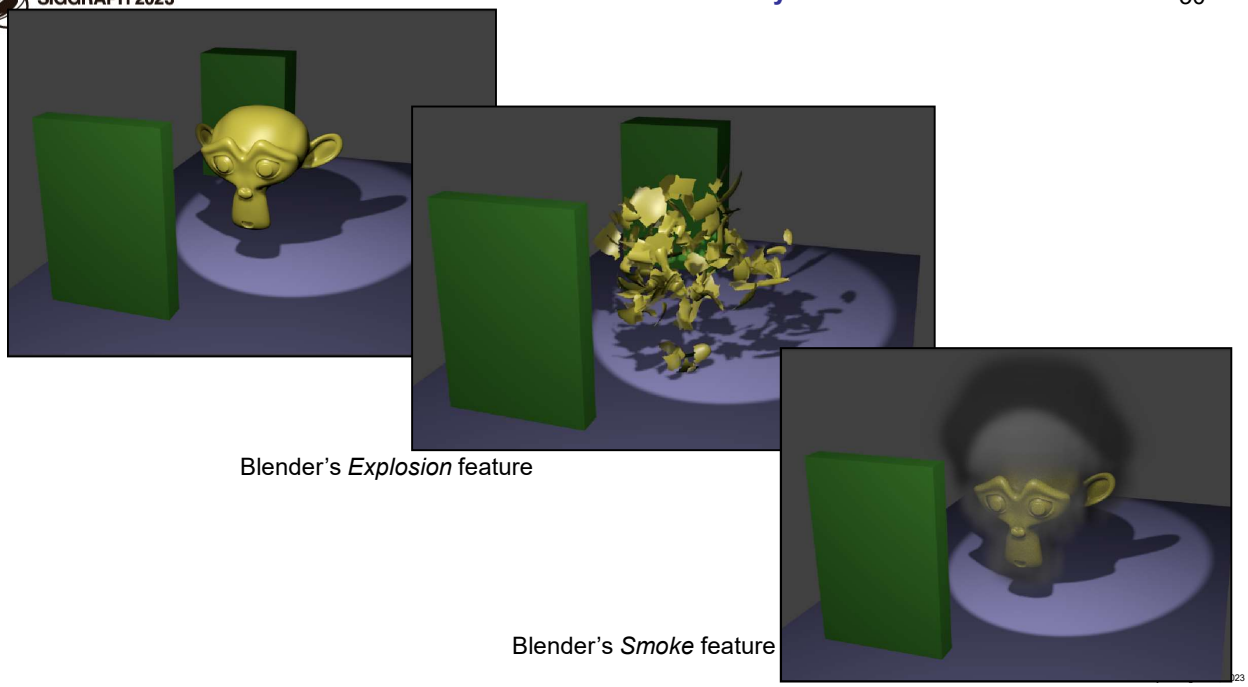


Difference

This is often called **Constructive Solid Geometry,** or **CSG**

Geometric Models can also be used for Physical Simulation

30

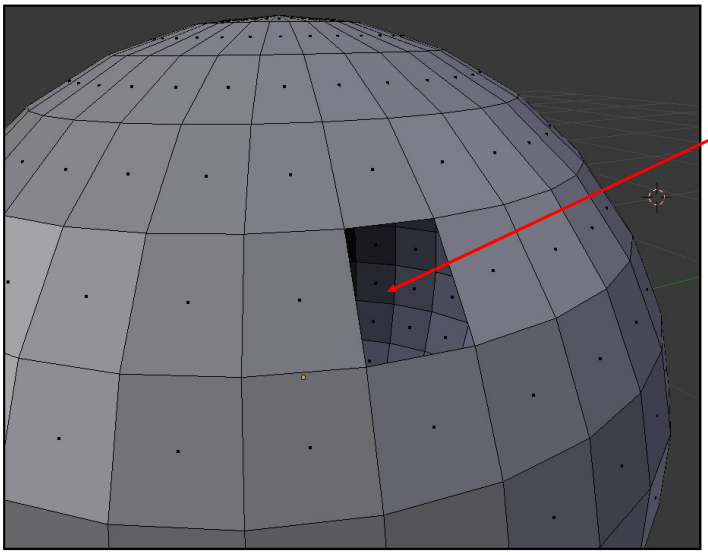


Blender's *Explosion* feature

Blender's *Smoke* feature

Object Modeling Rules for 3D Printing

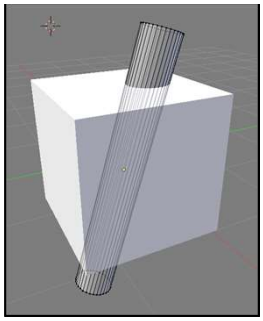
The object must be a legal solid. It must have a definite inside and a definite outside. It can't have any missing face pieces.



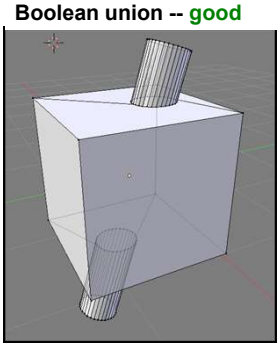
Object Modeling Rules for 3D Printing

Objects cannot pass through other objects. If you want two shapes together, do a CSG union on them so that they become one complete object.

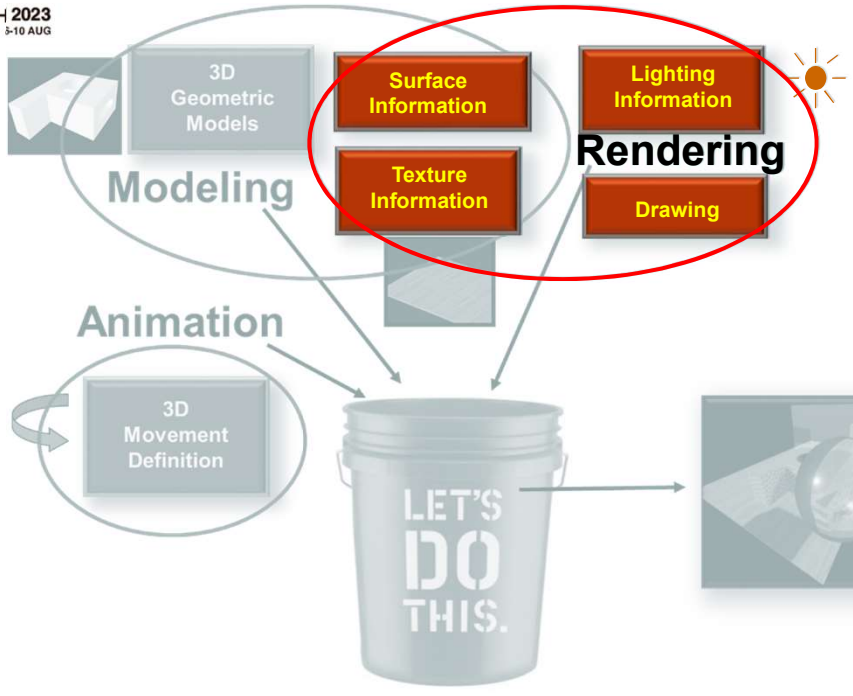
Overlapping in 3D -- bad



Boolean union -- good



Creating an image



Rendering

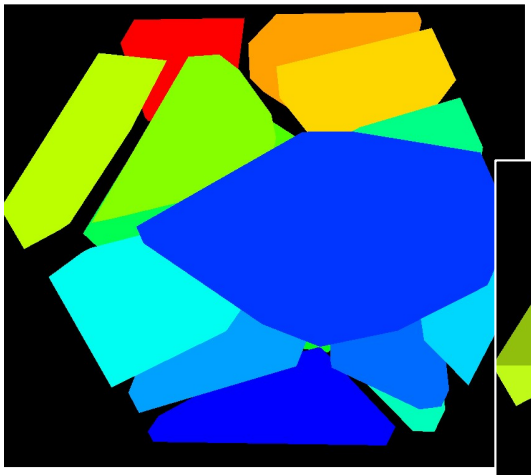
Rendering is the process of creating an image of geometric modes. Again, there are questions you need to ask first:

- Why am I doing this?
- How realistic do I want this image to be?
- How much compute time do I want this to take?
- Do I need to take lighting into account?
- Does the illumination need to be global or will local do?
- Do I need to create shadows?
- Do I need to create reflections and refractions?

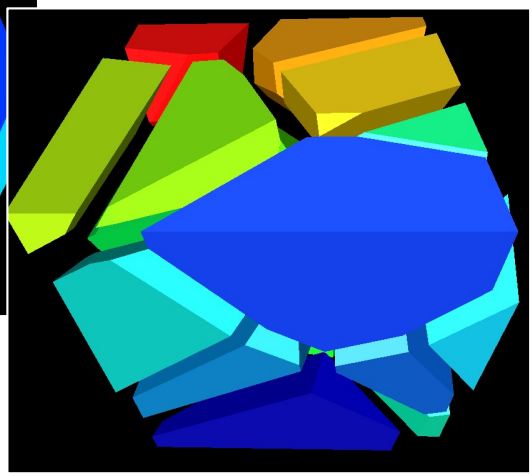
Want to experiment with a free rendering program?
Want some notes on how to get started?
http://cs.oregonstate.edu/~mjb/blender

Why Do We Care About Lighting?

No lighting

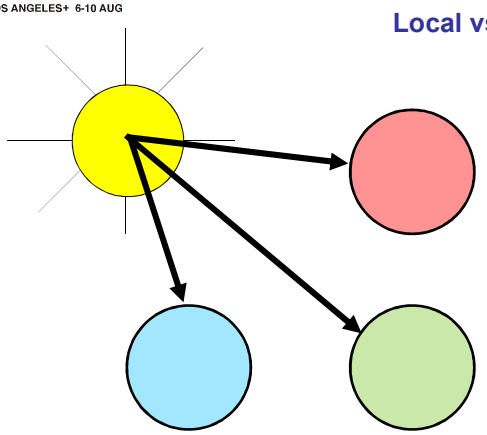


Lighting

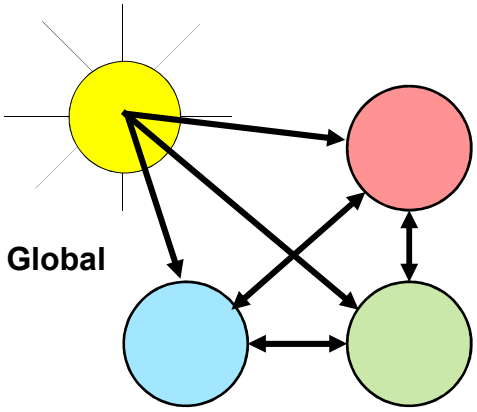


Lighting makes it possible to tell the difference between surfaces or parts of surfaces

Local vs. Global Illumination

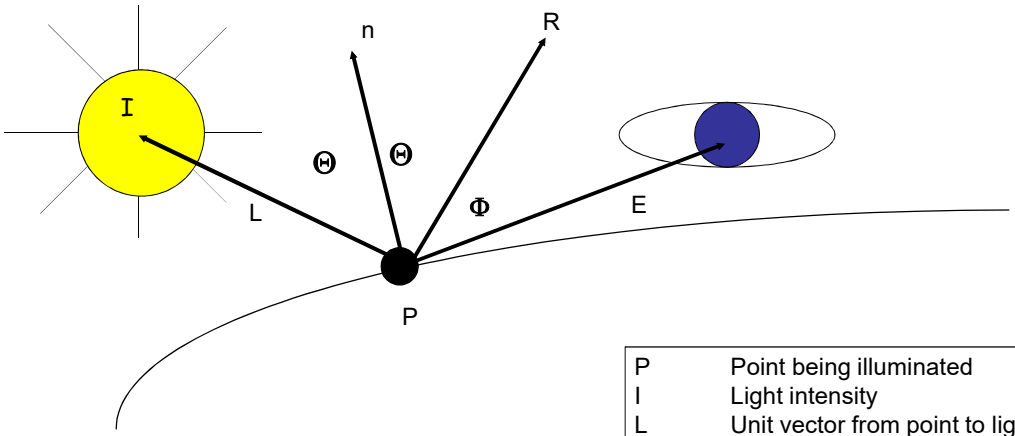


Local



Global

A Common type of Local Illumination: Ambient-Diffuse-Specular (ADS)



P	Point being illuminated
I	Light intensity
L	Unit vector from point to light
n	Unit vector surface normal
R	Perfect reflection unit vector
E	Unit vector to eye position

A Common type of Local Illumination: Ambient-Diffuse-Specular (ADS)

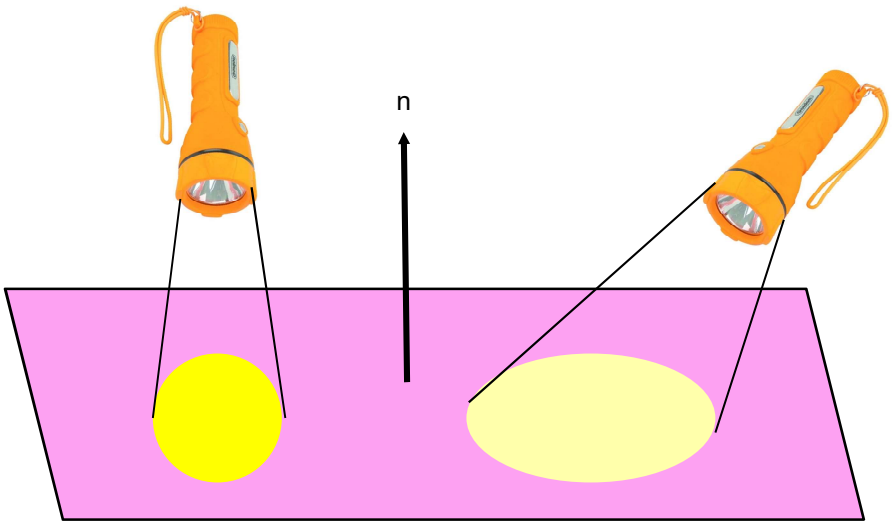
1. Ambient = a constant	Accounts for light bouncing "everywhere"
2. Diffuse = I*cosΘ	Accounts for the angle between the incoming light and the surface normal
3. Specular = I*cos^{s}ϕ	Accounts for the angle between the "perfect reflector" and the eye; also the exponent, S, accounts for surface shininess

Note that cosΘ is just the dot product between unit vectors L and n

Note that cosϕ is just the dot product between unit vectors R and E

Diffuse Lighting works because of spreading out the same amount of light energy across more surface area

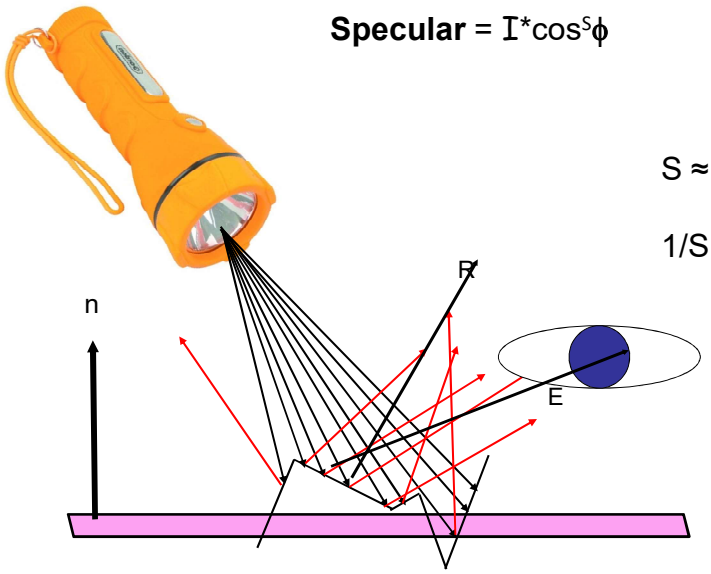
Diffuse = I*cosΘ



41

The Specular Lighting equation is a heuristic that approximates reflection from a rough surface

Specular = $I^*\cos^s\phi$



S ≈ "shininess"

1/S ≈ "roughness"

42

+

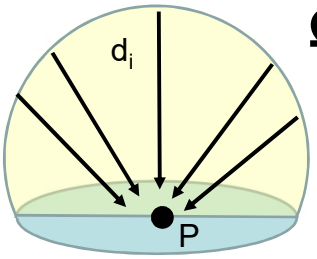
+

=

Put them all together!

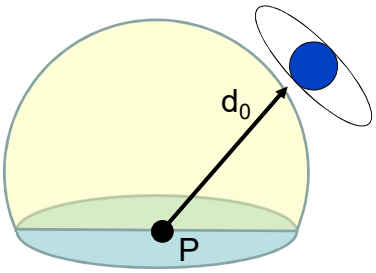
lighting.mp4

Global Illumination: The Rendering Equation



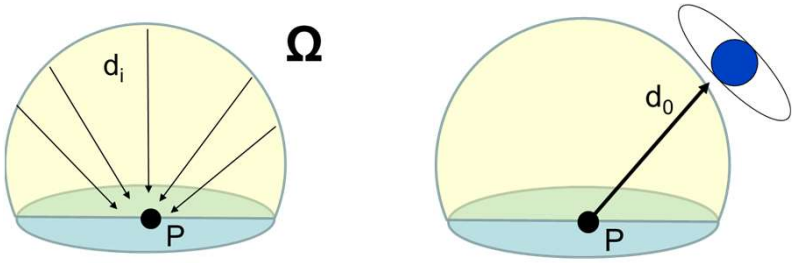
Light arriving at Point P from everywhere

Insert messy calculus equation here… ☺



Light departing from Point P in the direction that we are viewing the scene from

The Rendering Equation

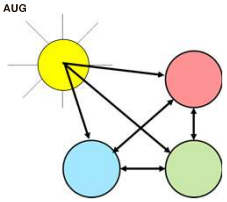


Insert messy calculus equation here… ☺

In plain language, the messy calculus describes a light balance:

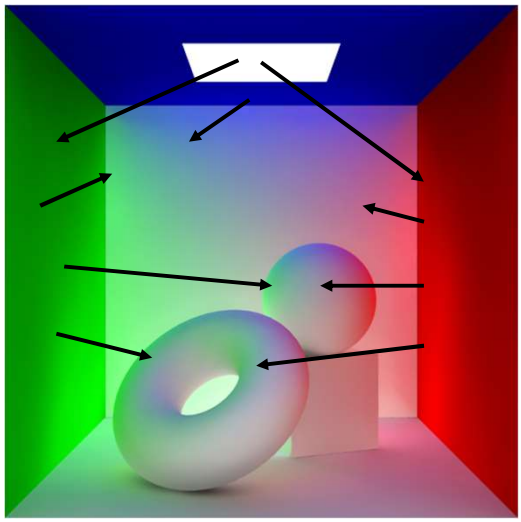
"The light shining from the point P towards your eye is the reflection of all the incoming light directed at the point P from all of the other objects in the scene."

The Lighting Equation at Work



- The left wall is green.
- The right wall is red.
- The back wall is white.
- The ceiling is blue with a light source in the middle of it.
- The objects sitting on the floor are white.

If the appearance of an object is also affected by the appearances of other objects, then you have **Global Illumination.**

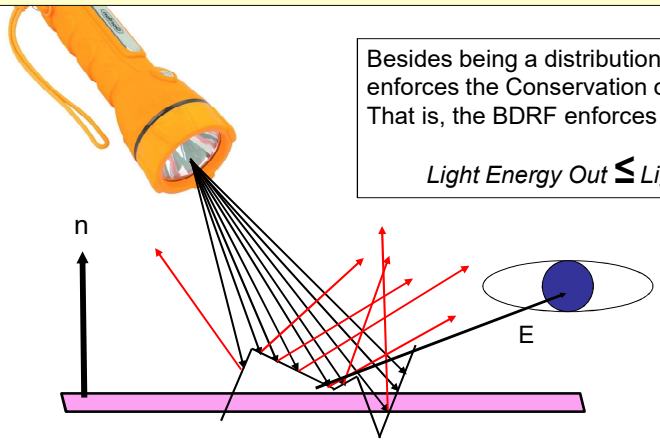


http://www.swardson.com/unm/tutorials/mentalRay3/

When light hits a surface, it bounces in particular ways depending on the angle and the material

This distribution of bounced light rays is called the **Bidirectional Reflectance Distribution Function**, or **BRDF**.

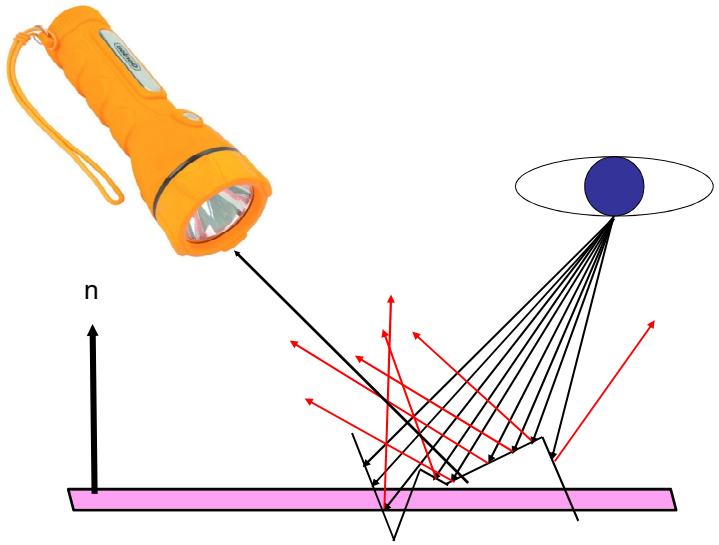
For a given material, the BRDF behavior of a light ray is a function of 4 variables: the 2 spherical coordinates of the incoming ray and the 2 spherical coordinates of the outgoing ray.



Besides being a distribution, the BDRF enforces the Conservation of Energy law. That is, the BDRF enforces

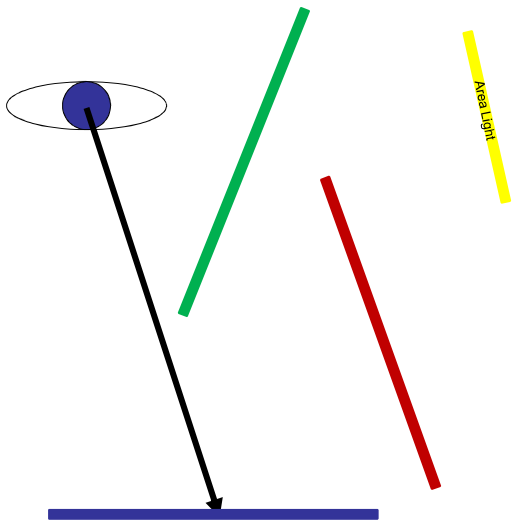
Light Energy Out ≤ *Light Energy In*

Usually it is easier to trace from the eye



Physically-Based Rendering

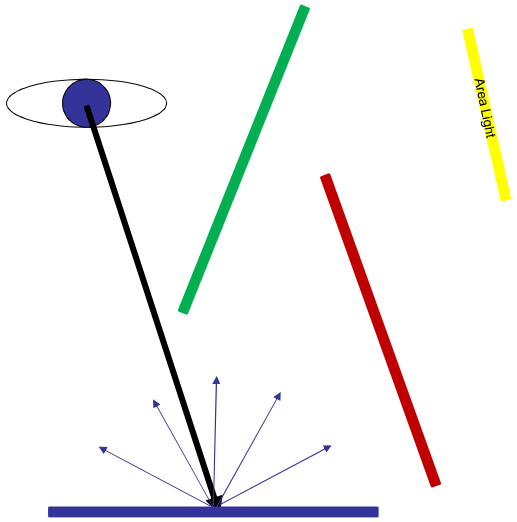
Let light can bounce around the scene, depending on how the different materials behave.



49

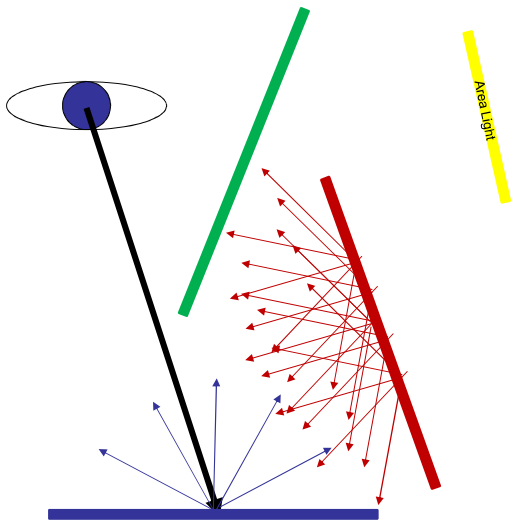
Physically-Based Rendering

Let light can bounce around the scene, depending on how the different materials behave.

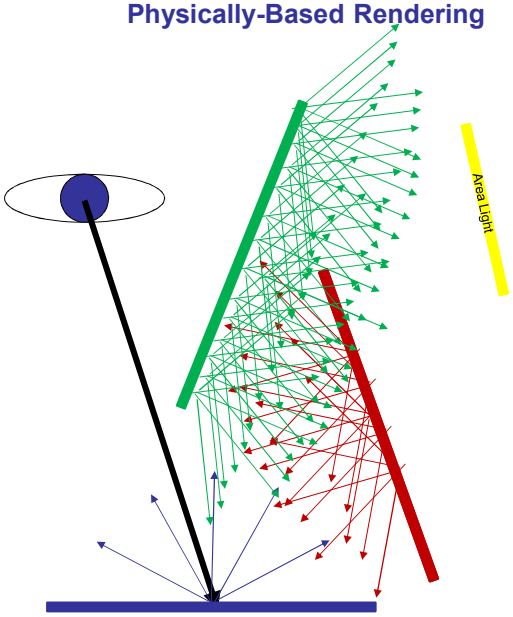


50

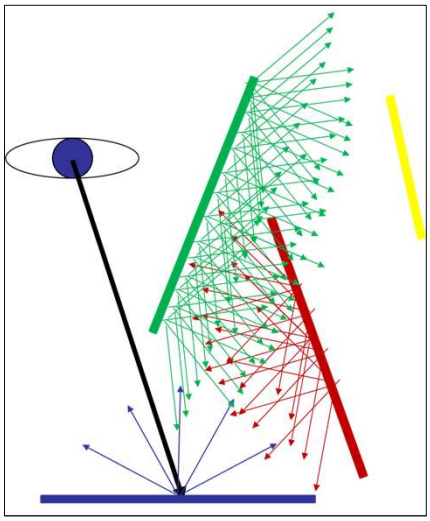
Physically-Based Rendering



Physically-Based Rendering



Physically-Based Rendering



Clearly this is capable of spawning an infinite number of rays. How do we handle this?

For a small-ish number of bounces, we can evenly distribute a collection of rays.

For lots of bounces, it's **Monte Carlo** simulation to the rescue!

$$LightGathered = \frac{\sum_0^{N-1} ResultOfRaysCastInRandomDirection}{N}$$

Recurse by applying this equation for all ray hits (yikes!)

Physically-Based Rendering using the Blender *Cycles* Renderer

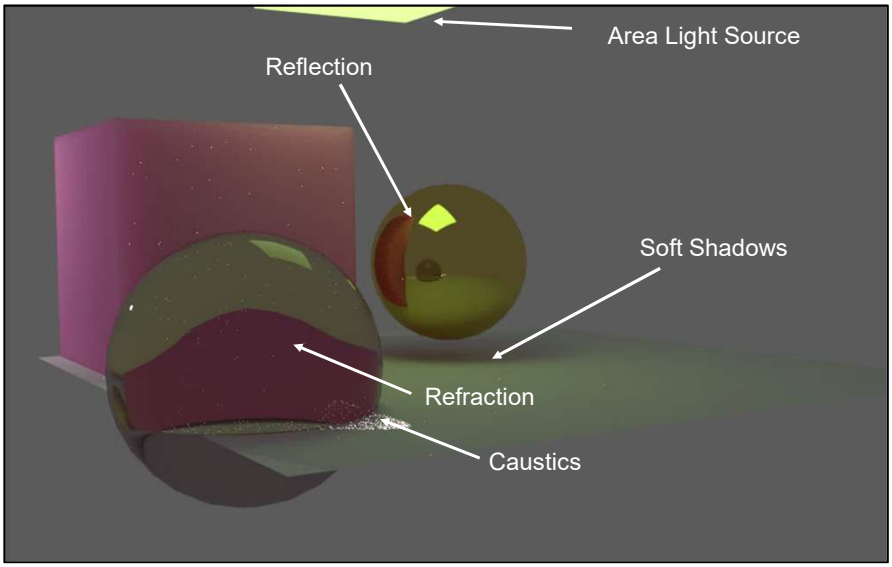


Image: Mike Bailey

Physically-Based Rendering using the Blender *Cycles* Renderer

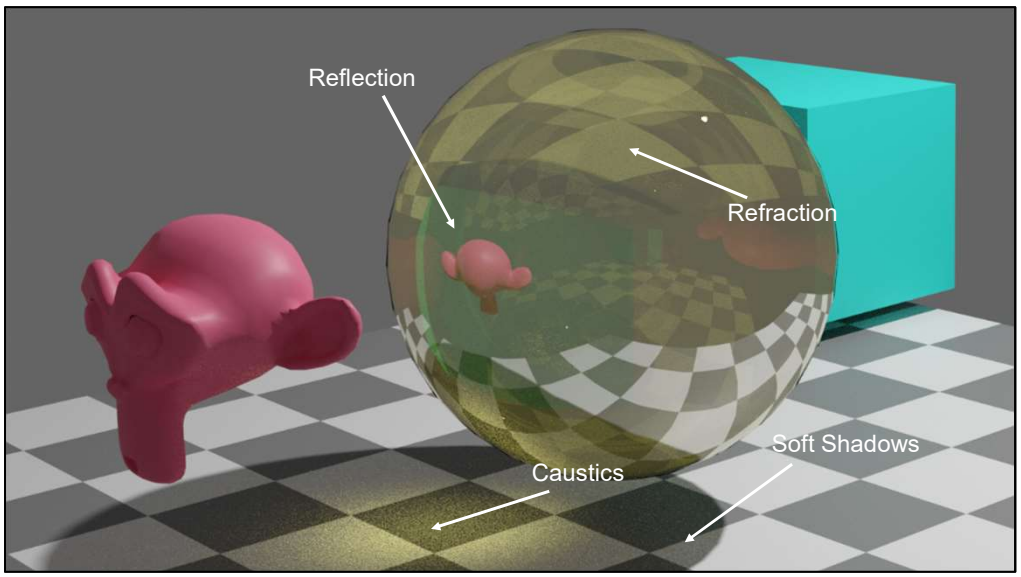
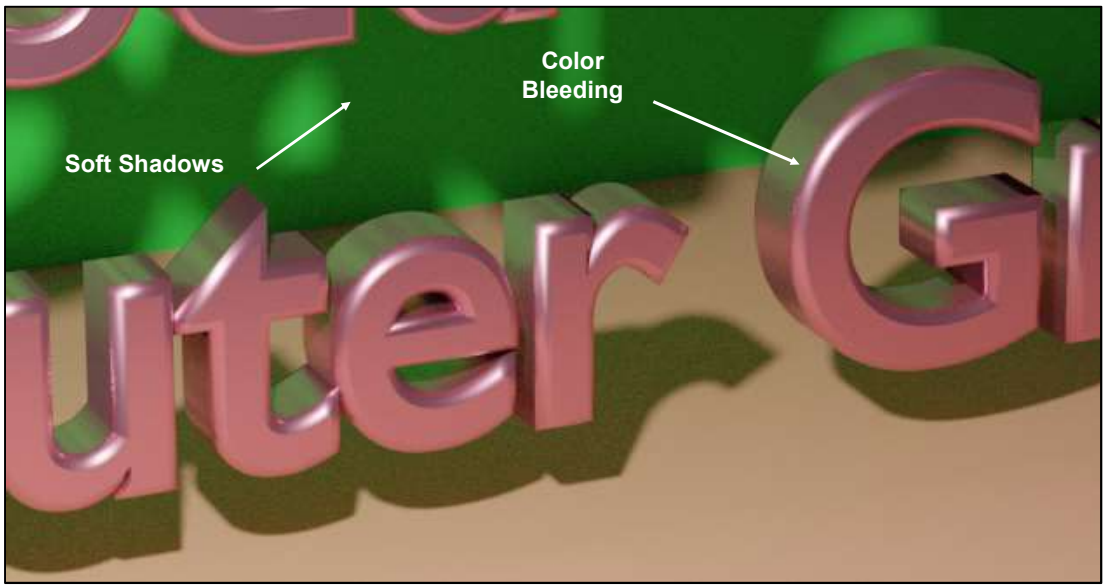


Image: Mike Bailey

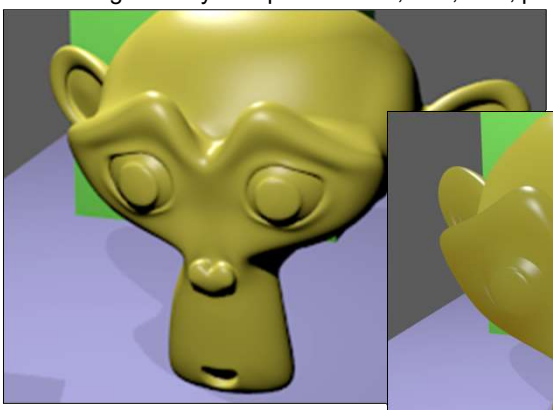
An Example from the Title Slide



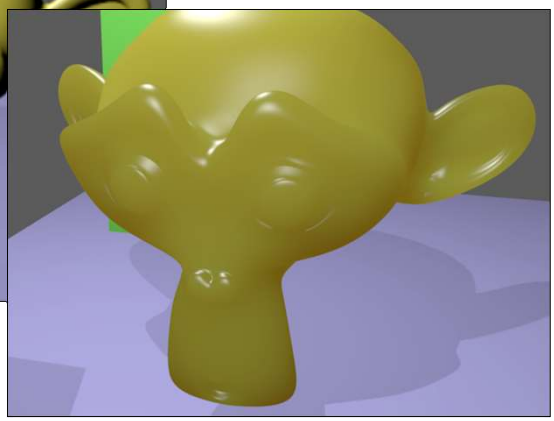
Subsurface Scattering

Subsurface Scattering models light bouncing around *within* an object before coming back out.

This is a good way to represent skin, wax, milk, paraffin, etc.



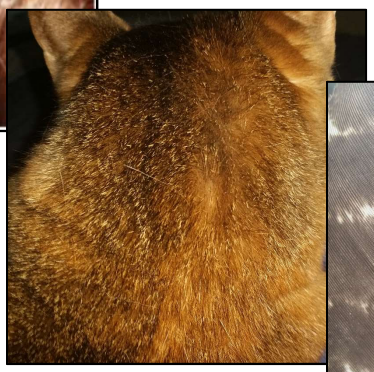
Without Subsurface Scattering



With Subsurface Scattering

Tricky Lighting Situations

Hair

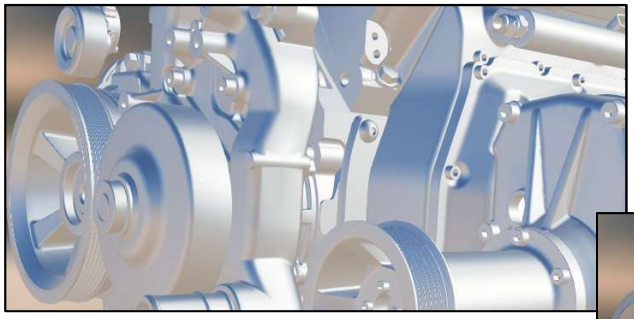


Fur

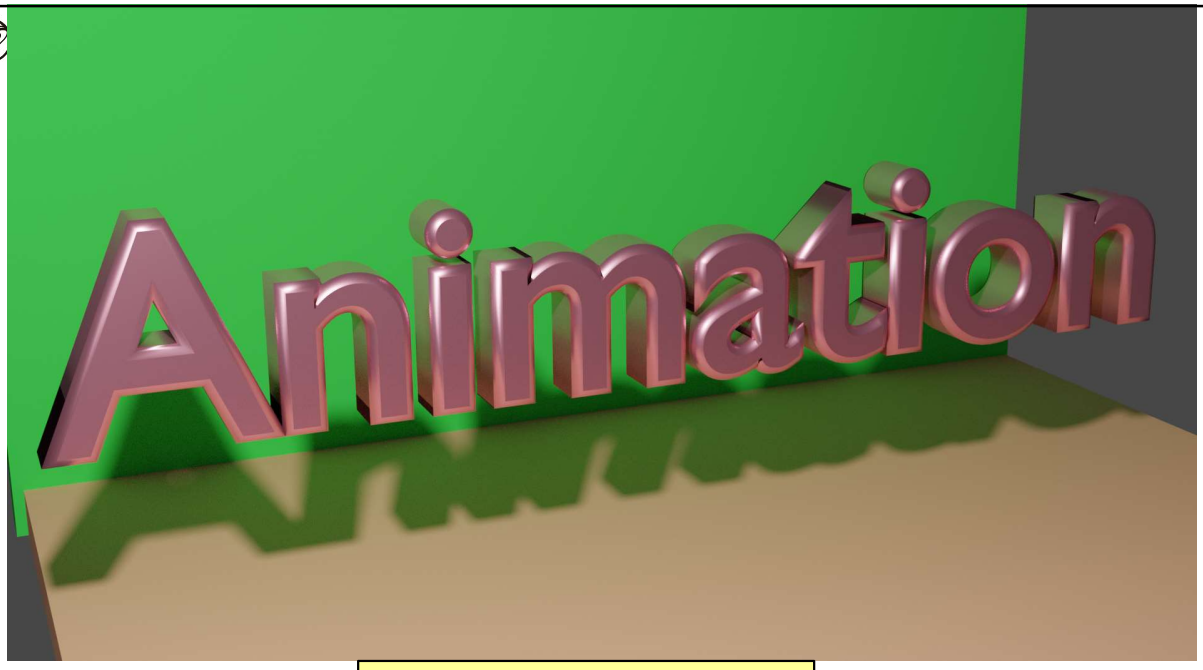
Feathers

Watch for these at the conference and in CG movies!

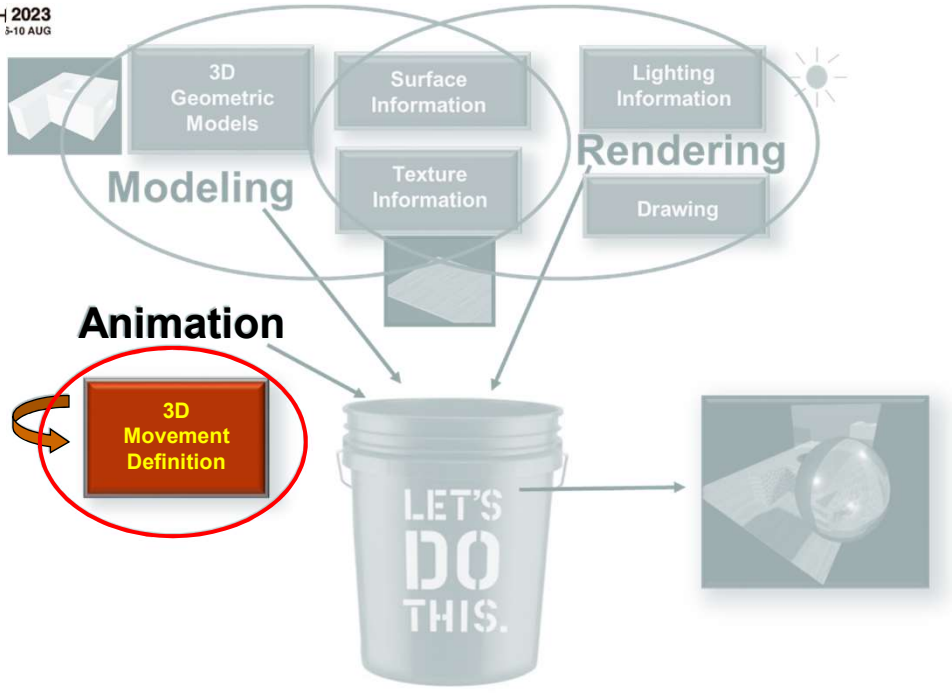
An Neat Global Illumination-ish Trick: Screen Space Ambient Occlusion (SSAO)



Kitware

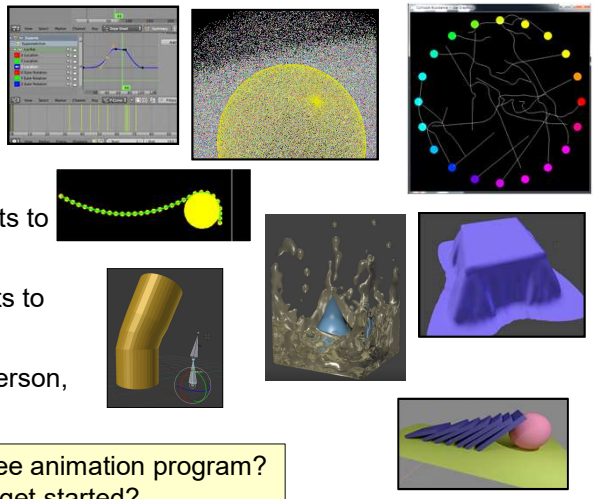


Creating the motion you want



Animation

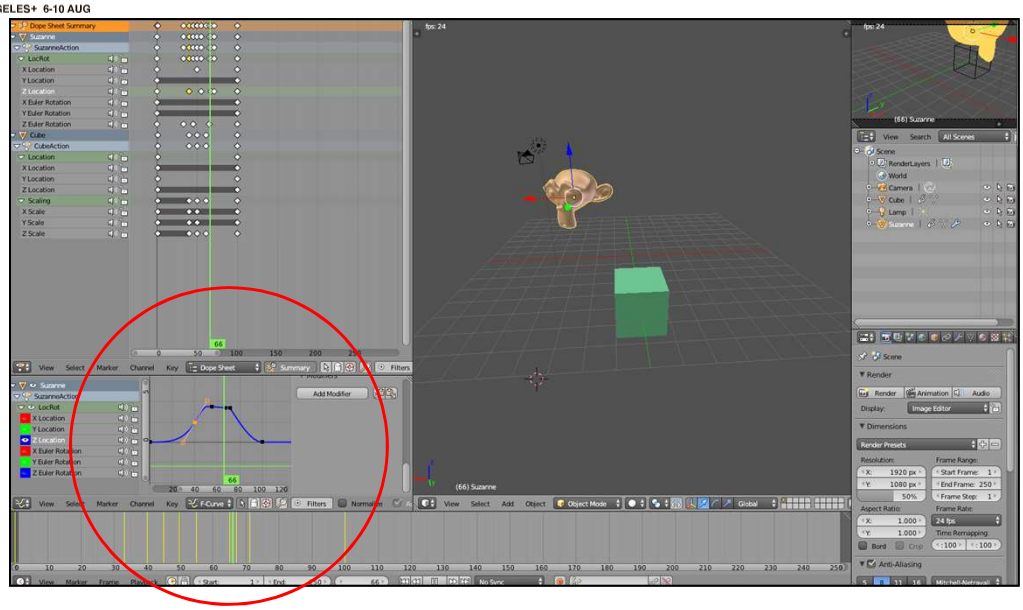
Rendering is the process of giving motion to your geometric modes. Again, there are questions you need to ask first:



- Why am I doing this?
- Do I want the animation to obey the real laws of physics?
- Am I willing to "fake" the physics to get the objects to *want* to move in a way that I tell it?
- Do I have specific key positions I want the objects to pass through no matter what?
- Do I want to simply record the motion of a real person, animal, etc., and then play it back?

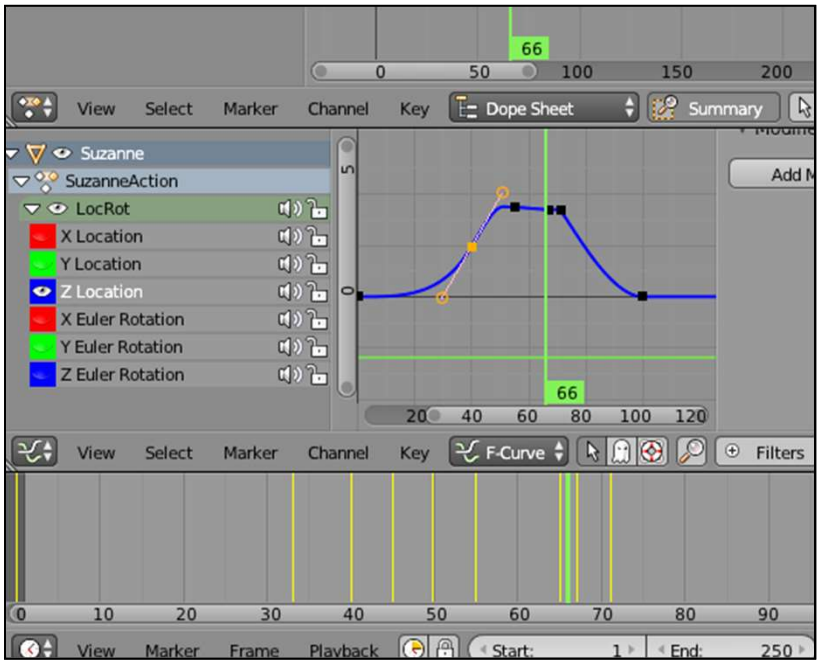
Want to experiment with a free animation program?
Want some notes on how to get started?
http://cs.oregonstate.edu/~mjb/blender

Keyframe Animation



Forcing the geometry to smoothly pass through key positions

Keyframe Animation



anim2.mp4

Sidebar: DIY KeyTime Animation

A simple C++ class can do it for you

class Keytimes:

```
void    AddTimeValue( float time, float value );
float   GetFirstTime( );
float   GetLastTime( );
int     GetNumKeytimes( );
float   GetValue( float time );
void    Init( );
void    PrintTimeValues( );
```

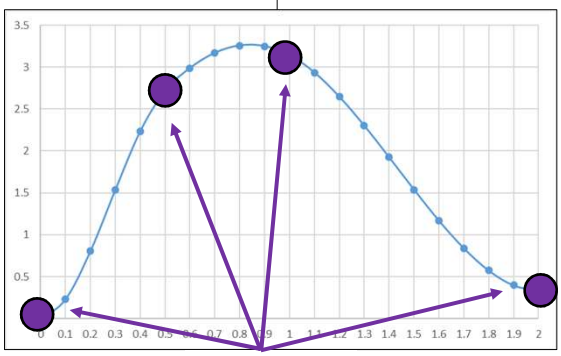
Find this code (keytime.h, keytime.cpp) on:
http://cs.oregonstate.edu/~mjb/whirlwind

DIY KeyTime Animation


```
Keytimes Xpos;

int
main( int argc, char *argv[ ] )
{
	Xpos.Init( );
	Xpos.AddTimeValue(  0.0,  0.000 );
	Xpos.AddTimeValue(  2.0,  0.333 );
	Xpos.AddTimeValue(  1.0,  3.142 );
	Xpos.AddTimeValue(  0.5,  2.718 );

	for( float t = 0.; t <= 2.0; t += 0.1 )		// just to show the example – we don't usually use a float in a for-loop
	{
		float v = Xpos.GetValue( t );
		fprintf( stderr, "%8.3f\t%8.3f\n", t, v );
	}
}
```


DIY KeyTime Animation


```
(  0.00,   0.000)
(  0.00,   0.000)  (  2.00,   0.333)
(  0.00,   0.000)  (  1.00,   3.142)  (  2.00,   0.333)
(  0.00,   0.000)  (  0.50,   2.718)  (  1.00,   3.142)  (  2.00,   0.333)
4 time-value pairs
Time runs from    0.000 to    2.000
   0.000         0.000
   0.100         0.232
   0.200         0.806
   0.300         1.535
   0.400         2.234
   0.500         2.718
   0.600         2.989
   0.700         3.170
   0.800         3.258
   0.900         3.250
   1.000         3.142
   1.100         2.935
   1.200         2.646
   1.300         2.302
   1.400         1.924
   1.500         1.539
   1.600         1.169
   1.700         0.840
   1.800         0.574
   1.900         0.397
   2.000         0.333
```



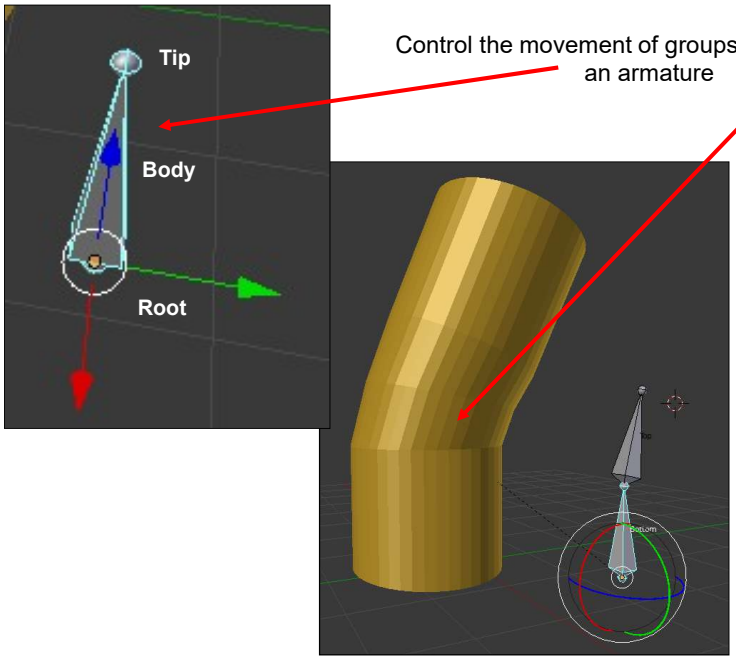
Key-Times

Using the System Clock for Timing in Display( )


```
#define MSEC		10000		// i.e., 10 seconds
Keytimes Xpos, Ypos, Zpos;
Keytimes ThetaX, ThetaY, ThetaZ;

. . .

if( AnimationIsOn )
{
	// # msec into the cycle ( 0 - MSEC-1 ):
	int msec = glutGet( GLUT_ELAPSED_TIME )  %  MSEC;

	// turn that into a time in seconds:
	float nowTime = (float)msec  / 1000.;
	glPushMatrix( );
		glTranslatef( Xpos.GetValue( nowTime ), Ypos.GetValue( nowTime ), Zpos.GetValue( nowTime )  );
		glRotatef(  ThetaX.GetValue( nowTime ),  1., 0., 0. );
		glRotatef(  ThetaY.GetValue( nowTime ),  0., 1., 0. );
		glRotatef(  ThetaZ.GetValue( nowTime ),  0., 0., 1. );
		<< draw the object >>
	glPopMatrix( );
}
```

Number of msec in the animation cycle

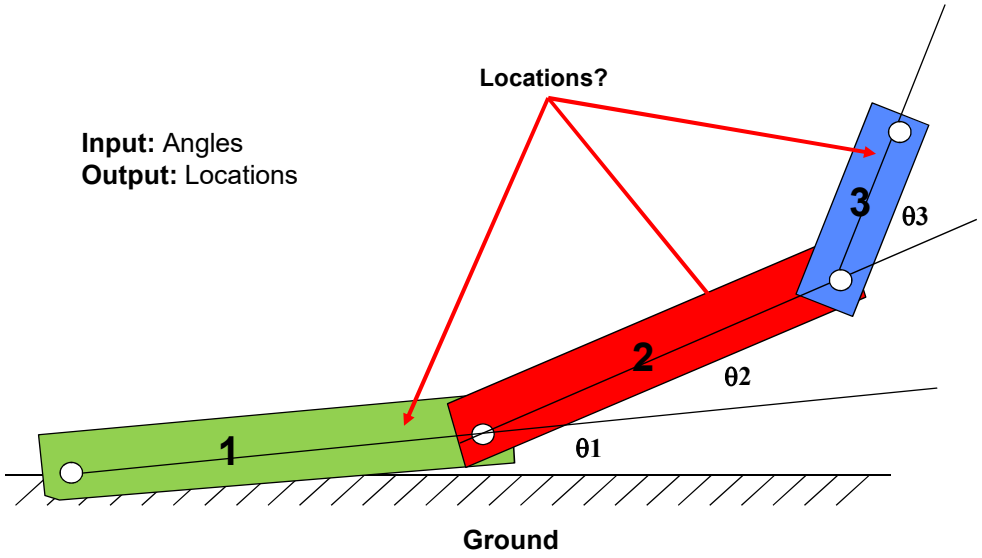
Rigging Animation



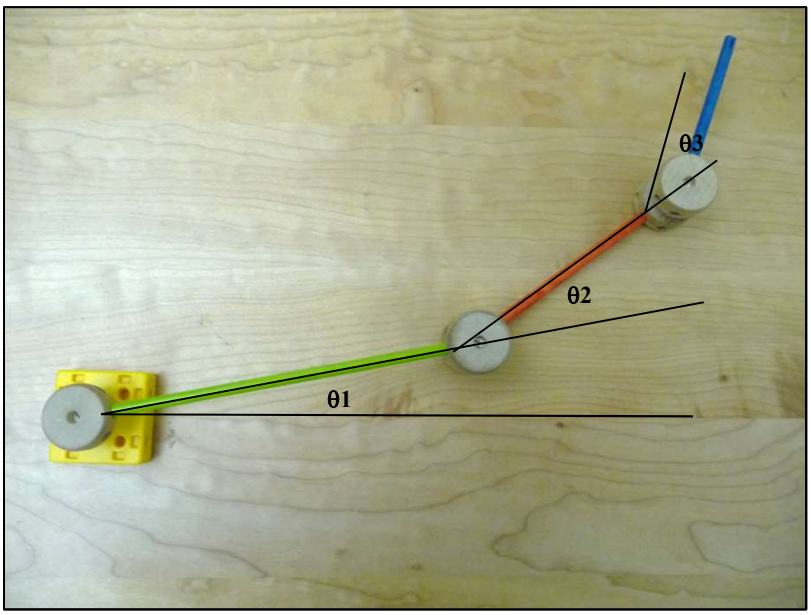
Control the movement of groups of vertices with an armature

Forward Kinematics: Transformation Hierarchies

Input: Angles
Output: Locations



Forward Kinematics:
Change Parameters – Things Move
(All Children Understand This)

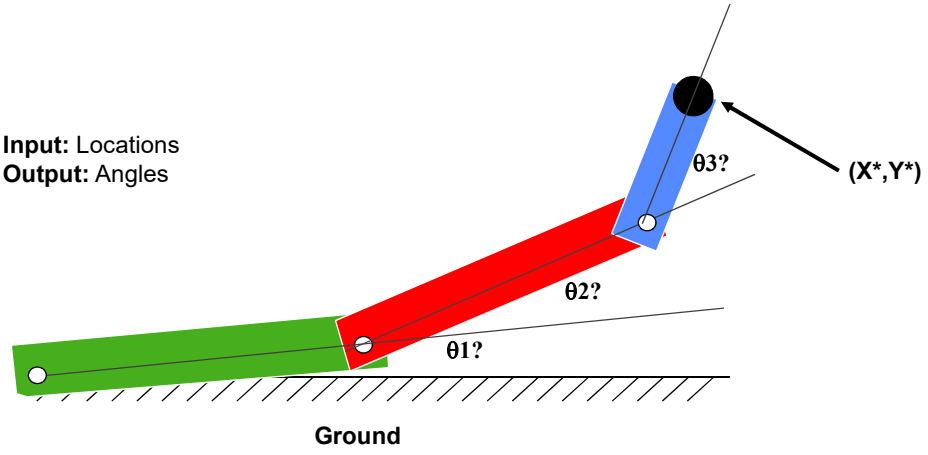


Inverse Kinematics

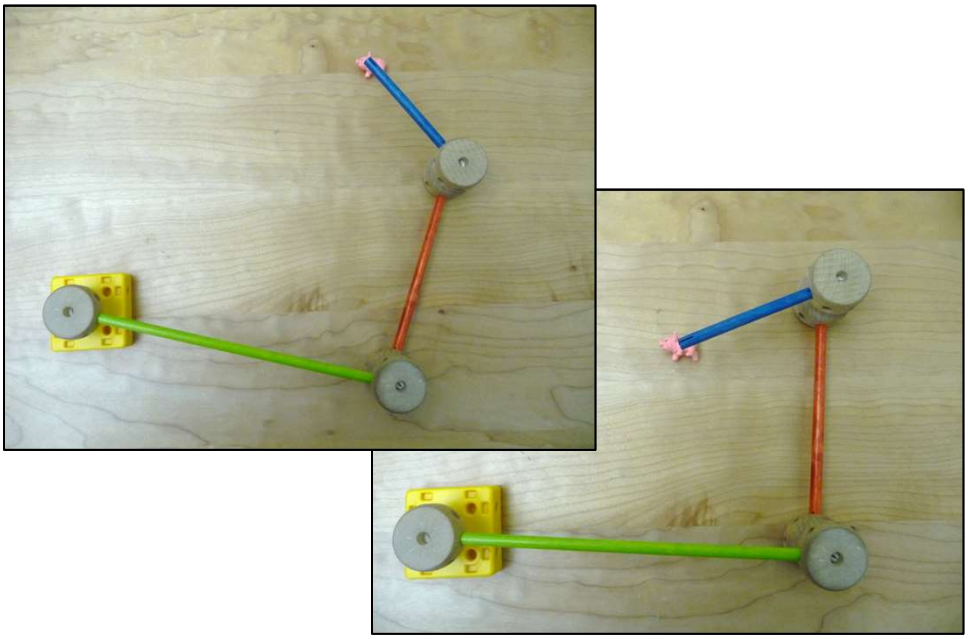
Forward Kinematics solves the problem "if I know the link transformation parameters, where are the links?".

Inverse Kinematics (IK) solves the problem "If I know where I want the end of the chain to be (X*,Y*), what transformation parameters will put it there?"

Input: Locations
Output: Angles



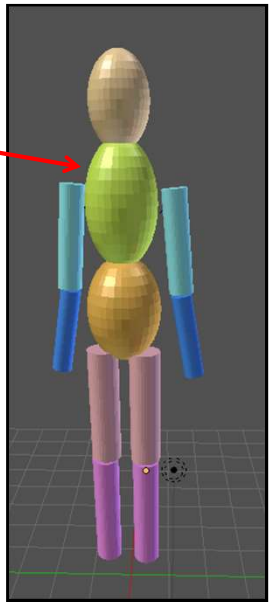
Inverse Kinematics (IK): Things Need to Move – What Parameters Will Make Them Do That?



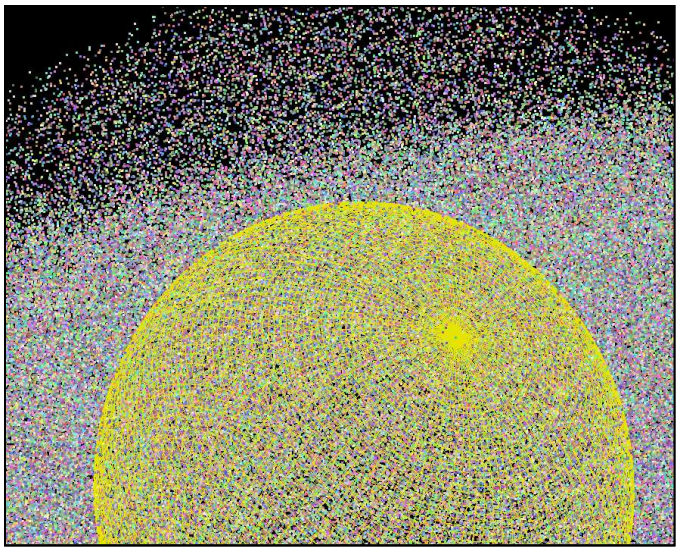
Animating a Human-ish Form

Start with this …

… and turn it into a kinematic model:



Particle Systems: A Cross Between Modeling and Animation?

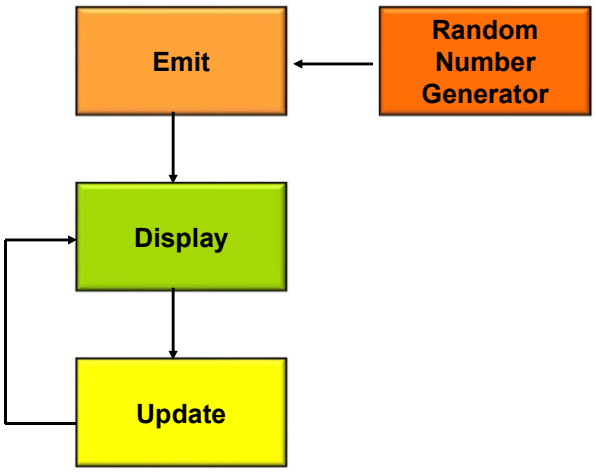


gpu_particles.mp4

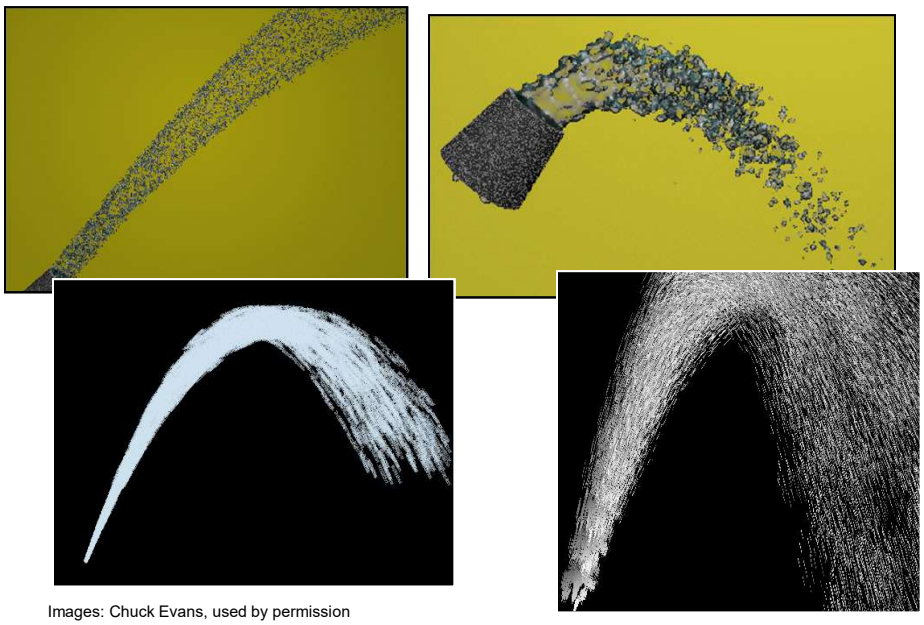
Check out this movie! These are particles animated on a GPU.

Particle Systems:
A Cross Between Modeling and Animation?

The basic process is:

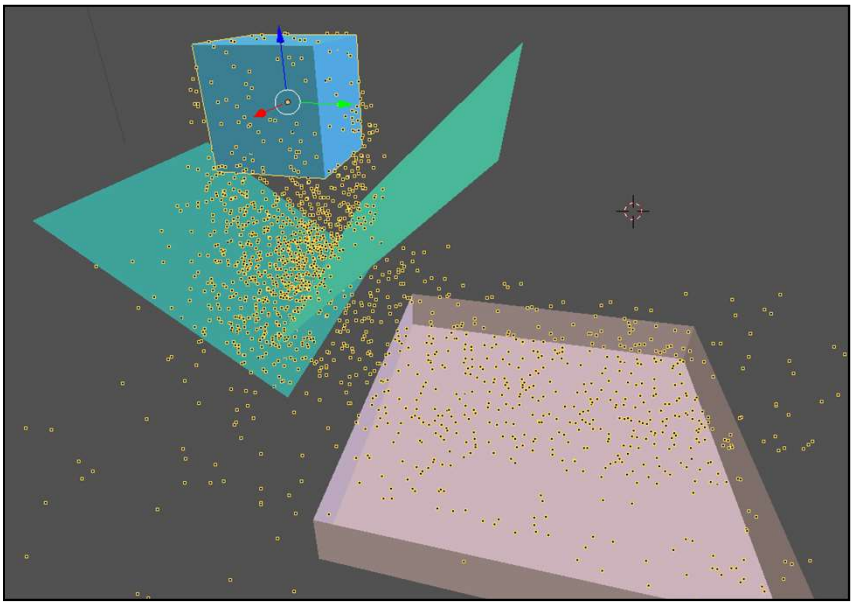


Particle Systems Examples



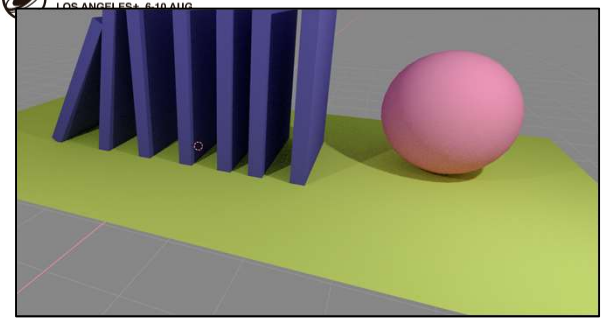
Images: Chuck Evans, used by permission

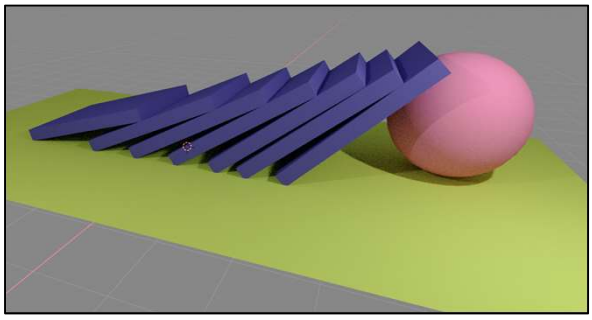
Particle Systems Examples



particles.mp4

Animating using Rigid-body Physics





Images: Mike Bailey

Newton's second law:

force = mass * acceleration

or:

acceleration = $\{\ddot{x}\}$ = force / mass

Newton's Second Law

dominos2.mp4

In order to make this work, you need to supply physical properties such as mass, center of mass, moment of inertia, coefficients of friction, coefficients of restitution, etc.

Animating using Fluid Physics

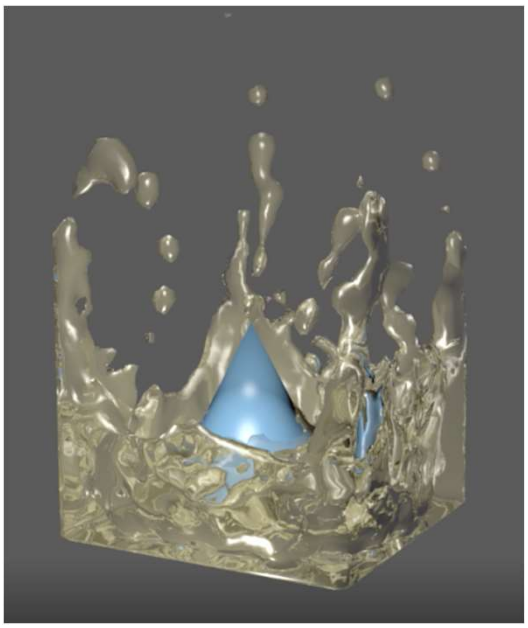
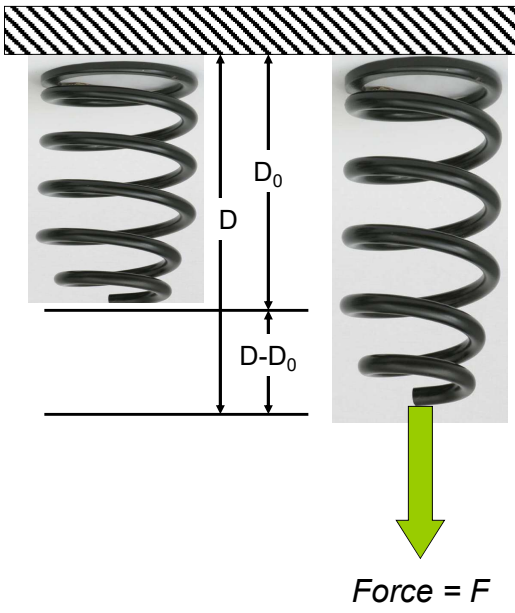


Image: Mike Bailey

fluid.avi

Animating using Spring Physics



D_0 = unloaded spring length

$$(D - D_0) = \frac{F}{k}$$

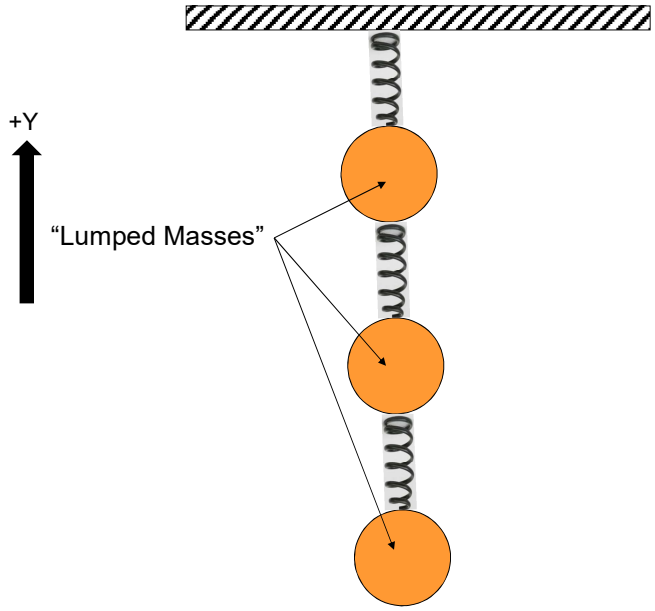
k = ***spring stiffness*** in Newtons/meter or pounds/inch

Or, if you know the displacement, the force exerted by the spring is:

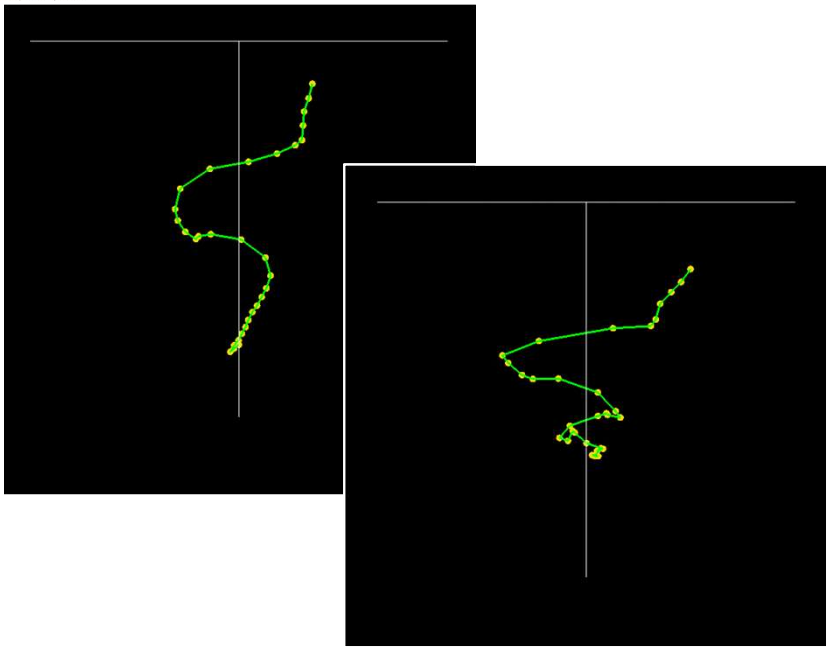
$$F = k\left(D - D_0\right)$$

This is known as **Hooke's Law**

Animating using the Physics of a Mesh of Springs

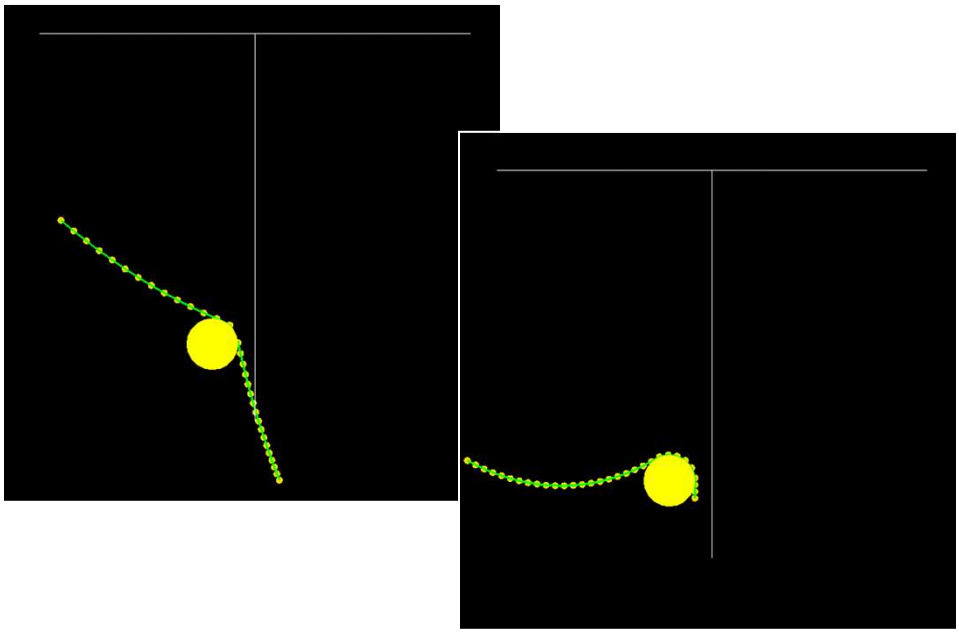


Simulating a Bouncy Chain

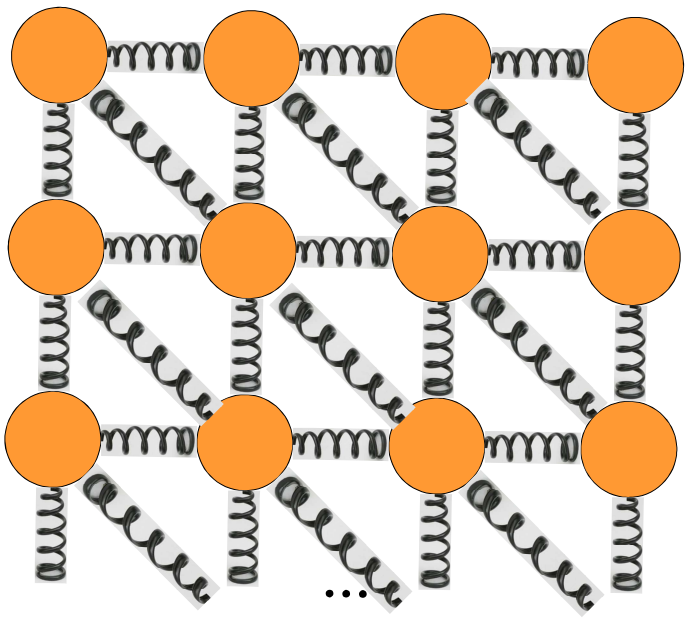


chain.mp4

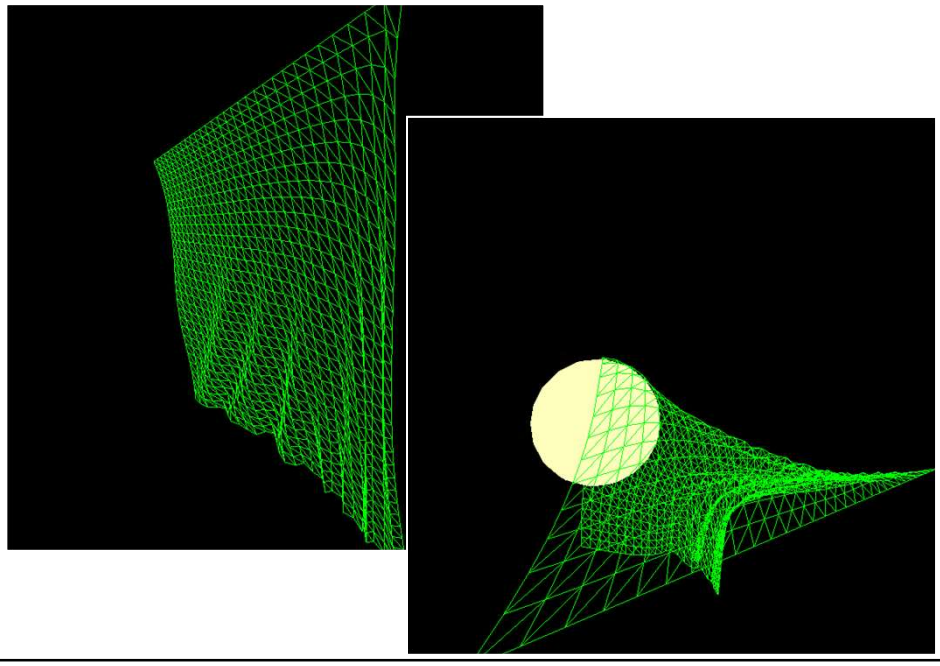
Placing a Physical Barrier in the Scene



Animating Cloth



Cloth Examples



Cloth Example

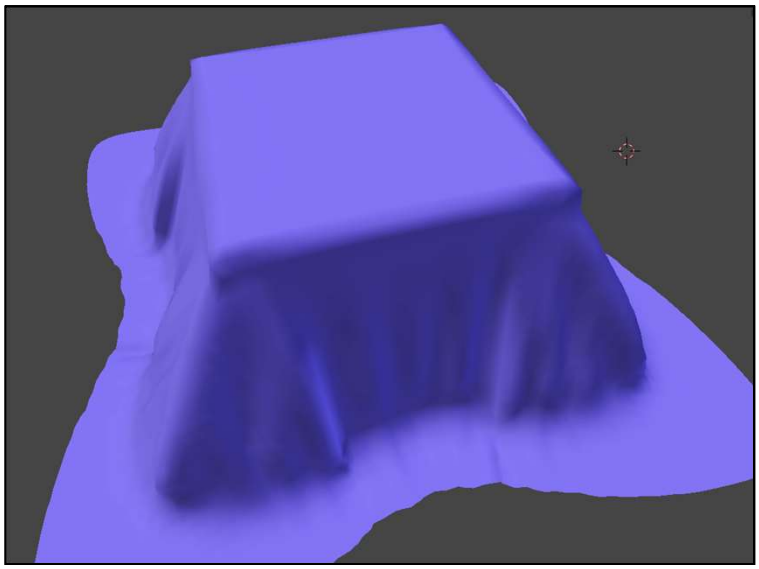


Image: Mike Bailey

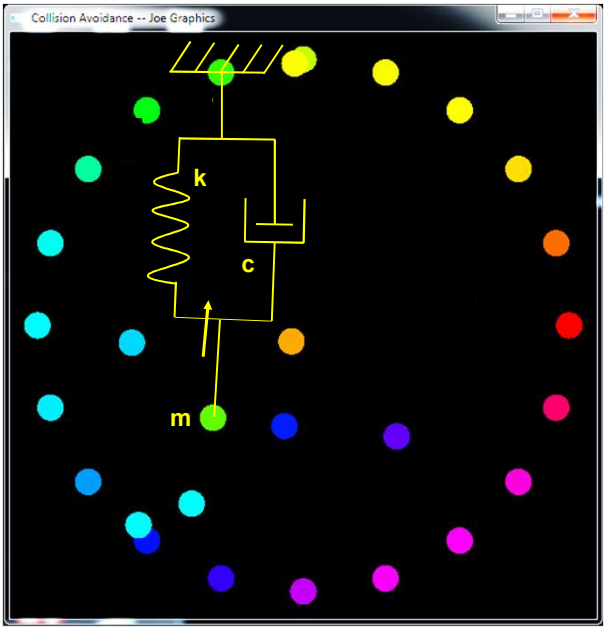
cloth.mp4

Functional Animation – "Fake Physics"

Photo: Mike Bailey

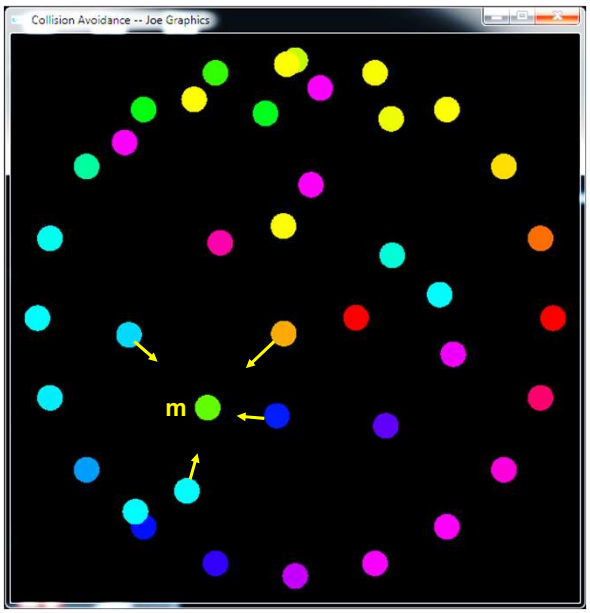
The Challenge: animate a collection of objects, each trying to move to a target, but without colliding with each other

Functional Animation:
Make the Object *Want* to Move Towards a Goal Position . . .



$$m\ddot{x}+c\dot{x}+kx=0$$

Functional Animation:
. . . While Making it *Want* to Keep Away from all other Objects



$$m\ddot{x} = \sum F_{repulsive}$$

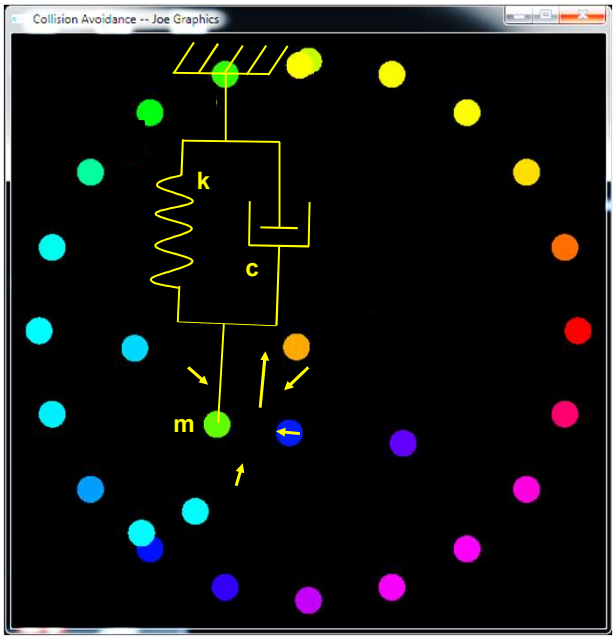
Repulsion Coefficient

$$F_{repulsive} = \frac{C_{repulse}}{d^{Power}}$$

Distance between the boundaries of the 2 bodies

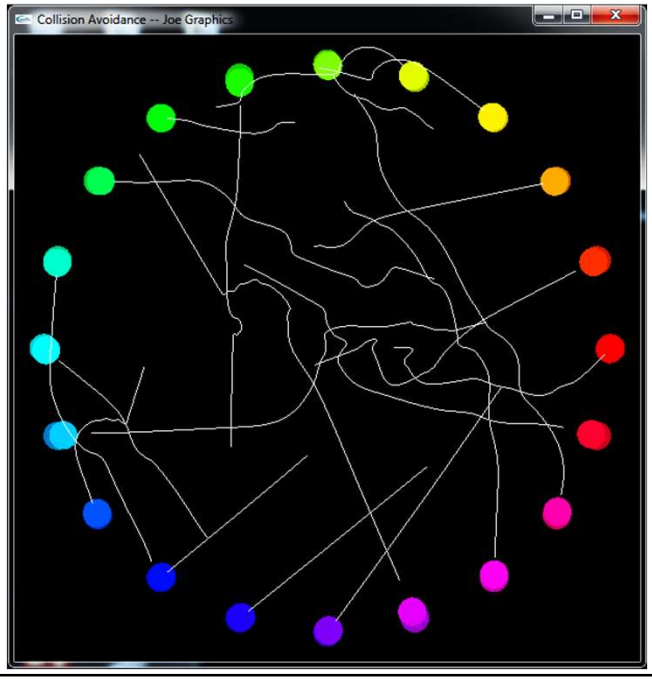
Repulsion Exponent

Total Goal – Make the Free Body Move Towards its Final Position While Being Repelled by the Other Bodies



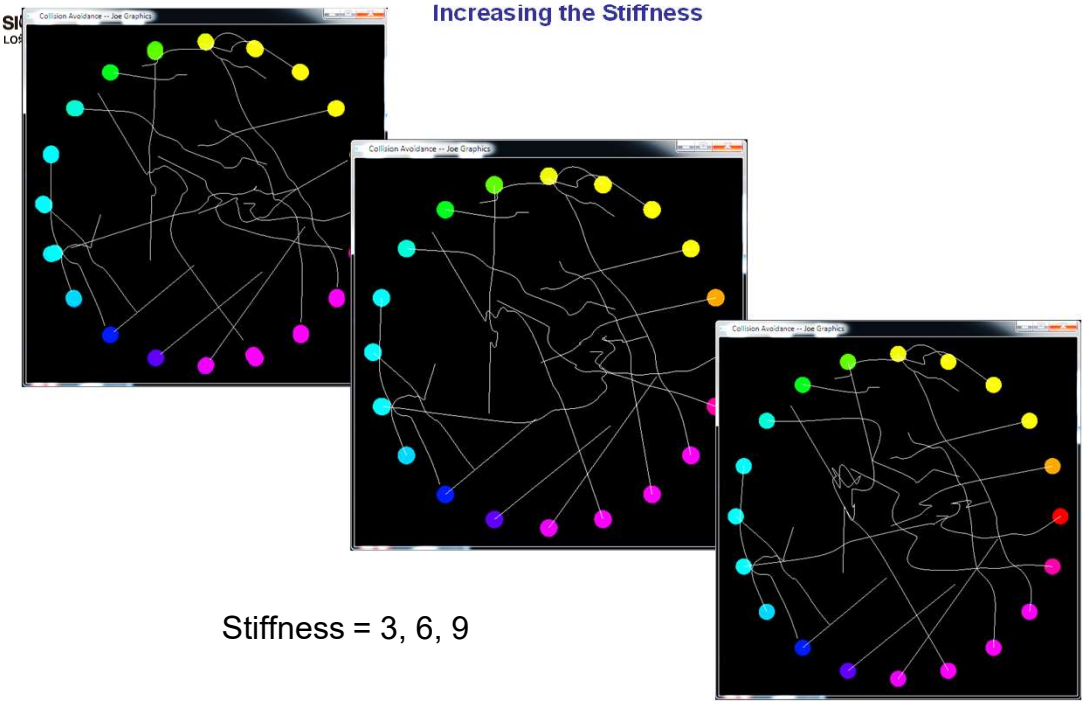
$$m\ddot{x}+c\dot{x}+kx = \sum F = \sum F_{repulsive}$$

Functional Animation



avoid.mp4

Increasing the Stiffness

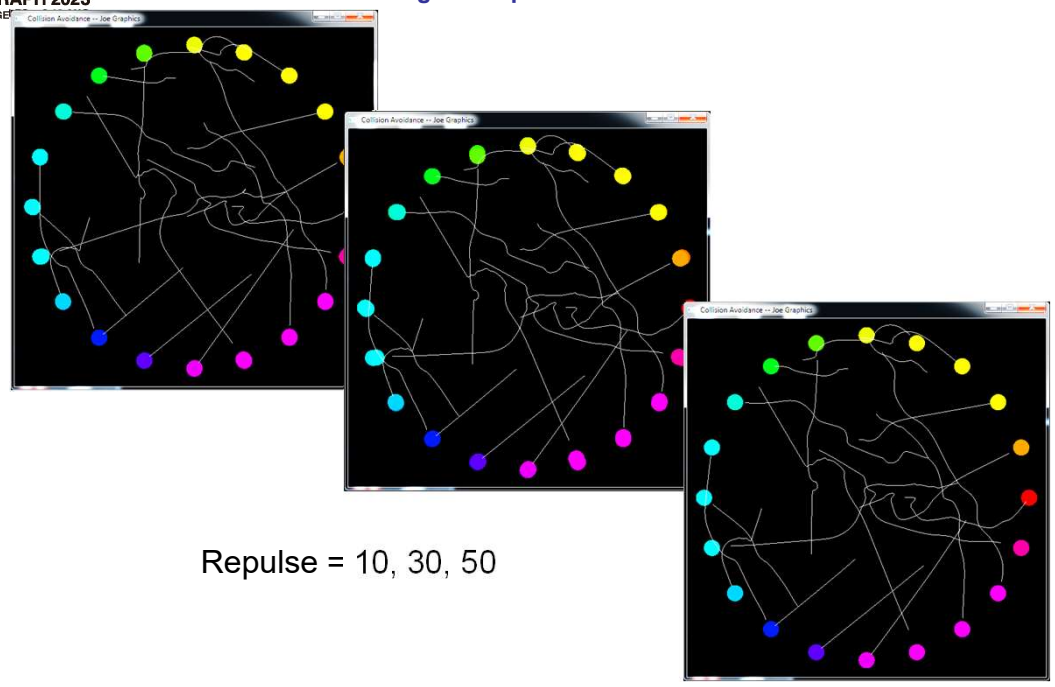


Stiffness = 3, 6, 9

SIGGRAPH 2023
LOS ANGE

Increasing the Repulsion Coefficient

93



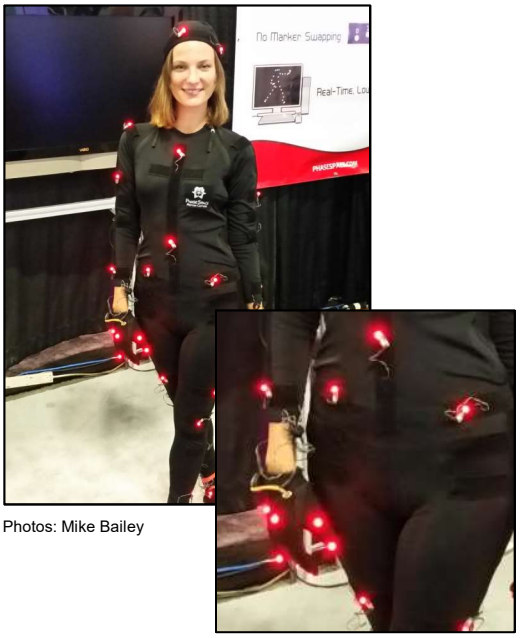
Repulse = 10, 30, 50

mjb – August 14, 2023

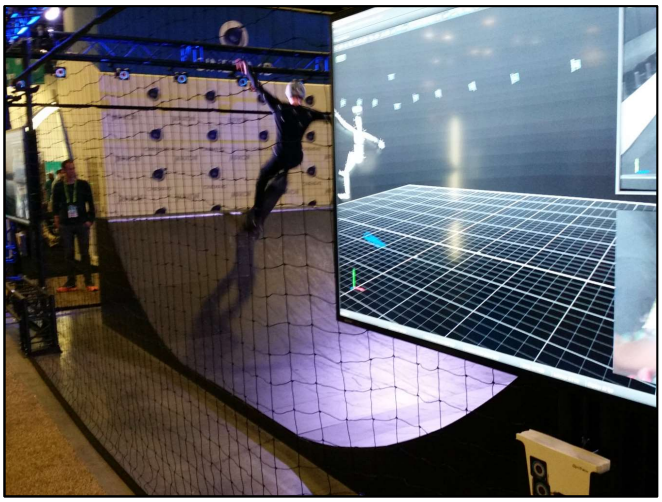
SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

Motion Capture ("MoCap") as an Input for Animation

94



Photos: Mike Bailey



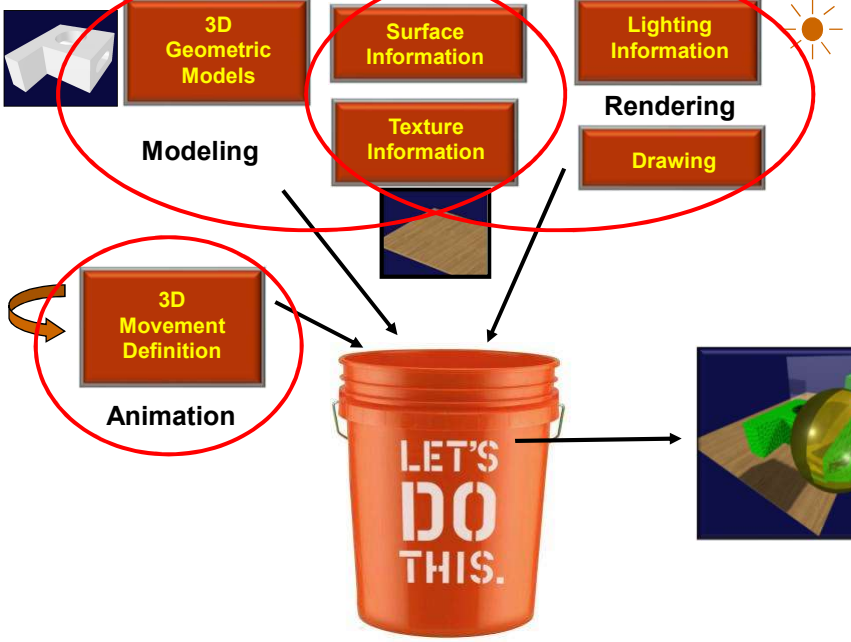
mjb – August 14, 2023

Even Animals can be MoCapped

https://www.youtube.com/watch?v=zyq_LQrHpoo

Photo courtesy of: DIGIC Services' Mocap Studio, used by permission

Summary



Conclusions !

- SIGGRAPH moments will never come again. Well, this is usually true, but through magic of the 2023 videos, they might. *But, be aware of what is going to be archived and what isn't.* And, if it is to be archived, how long will you have access to it?
- Especially take advantage of the not-to-be-archived or not-to-be-archived-for-very-long events because you cannot re-live them forever.
- Combine what you have just learned here with what else you learn at the conference and *relate them to your career and life goals*.
- Have fun doing it!

http://cs.oregonstate.edu/~mjb/whirlwind

99

Check out the *More Information* Document to be found at:

Where to Find More Information about Computer Graphics and Related Topics

Mike Bailey
Oregon State University

1. References

1.1 General Computer Graphics

SIGGRAPH Online Bibliography Database:
http://www.siggraph.org/learn/computer-graphics-bibliography-database

Edward Angel and Dave Shreiner, *Interactive Computer Graphics: A Top-down Approach with OpenGL*, 6th Edition, Addison-Wesley, 2011.

Francis Hill and Stephen Kelley, *Computer Graphics Using OpenGL*, 3rd Edition, Prentice Hall, 2006.

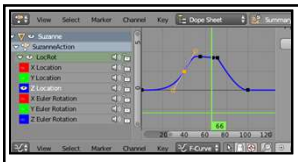
Steve Cunningham, *Computer Graphics: Programming in OpenGL for Visual Communication*, Prentice-Hall, 2007

Alan Watt, *3D Comput*

Peter Shirley, *Fundamentals* of Computer Graphics, 2nd Edition, AK Peters, 2005.

Andrew Glassner, *Graphics Gems*, Academic Press, 1990.

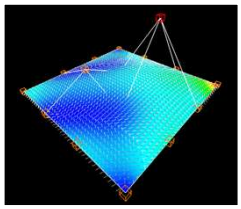
http://cs.oregonstate.edu/~mjb/whirlwind

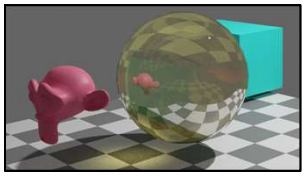


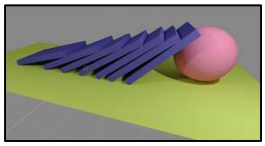
100

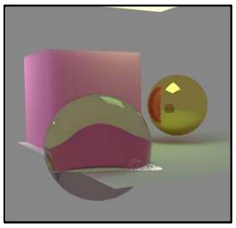
A Whirlwind Introduction to Computer Graphics

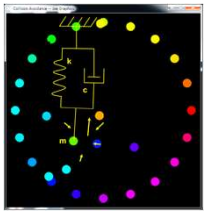
Mike Bailey
mjb@cs.oregonstate.edu











http://cs.oregonstate.edu/~mjb/whirlwind