IT UNIVERSITY OF COPENHAGEN

Second Year Project

Advanced scene building

Shape manipulation

Affine Transformations

- Scaling
- Rotating
- Shearing
- Translation (moving)
- Cloning

Constructive Solid Geometry

- Union
- Intersection
- Subtraction

But First....

Are there any questions from last lecture?

Information

- Patrick is teaching the next two weeks
- All projects must be stored on Github and Patrick, myself and the TAs must be given read-only access. You may use standard Github, but the repositories must be private
- The API is up

Why affine transformations

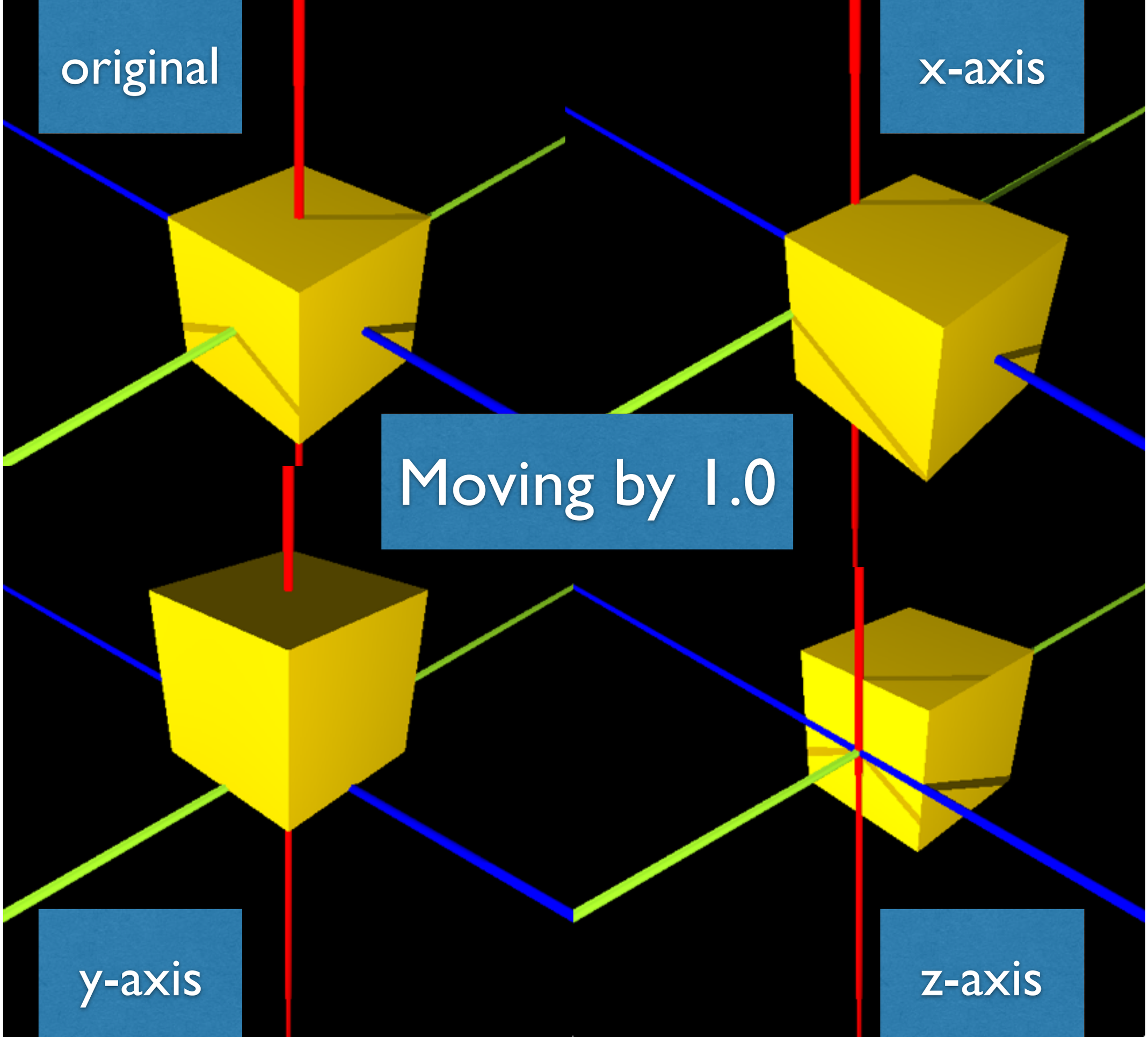
- Many objects are expensive to load (triangle meshes, implicit surfaces)
- Some hit functions are much simpler if you can assume some basic properties (axis alignment, centred at origo, radius or sides of 1 unit)
- Gives you fine-grained control over your scene

Test suite

Choose a shape, an image resolution and a save location, and the test suite will generate a set of files that test all of your transformations on the chosen shape

Translation

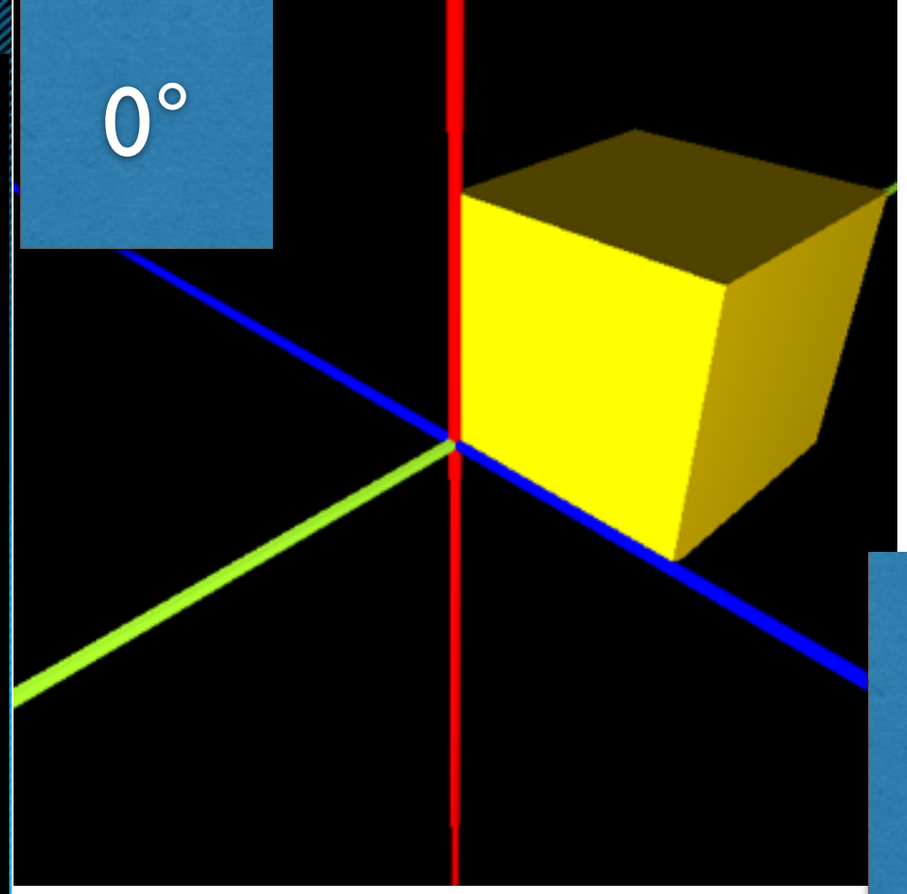
- Translation is just a fancy word for moving
- We can move shapes along all axes
- In the following pictures x-axis is in blue, y-axis is in yellow and z-axis is in red

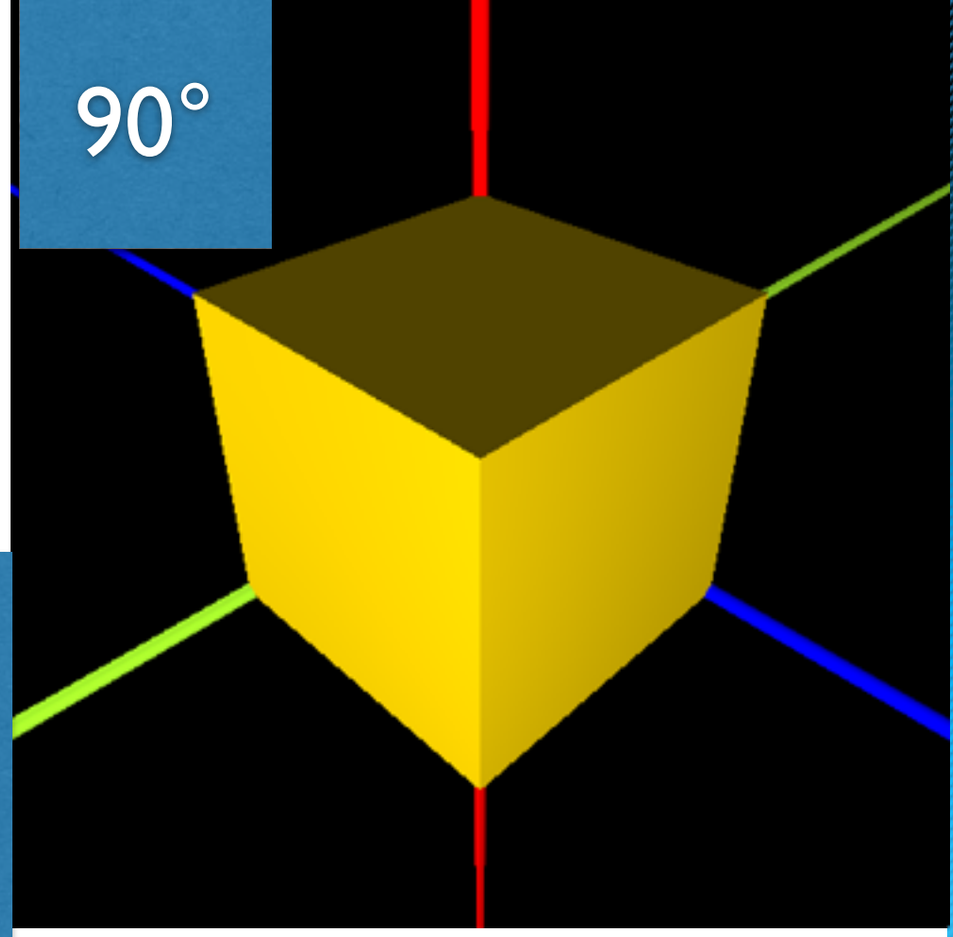

original
x-axis
Moving by 1.0
y-axis
z-axis

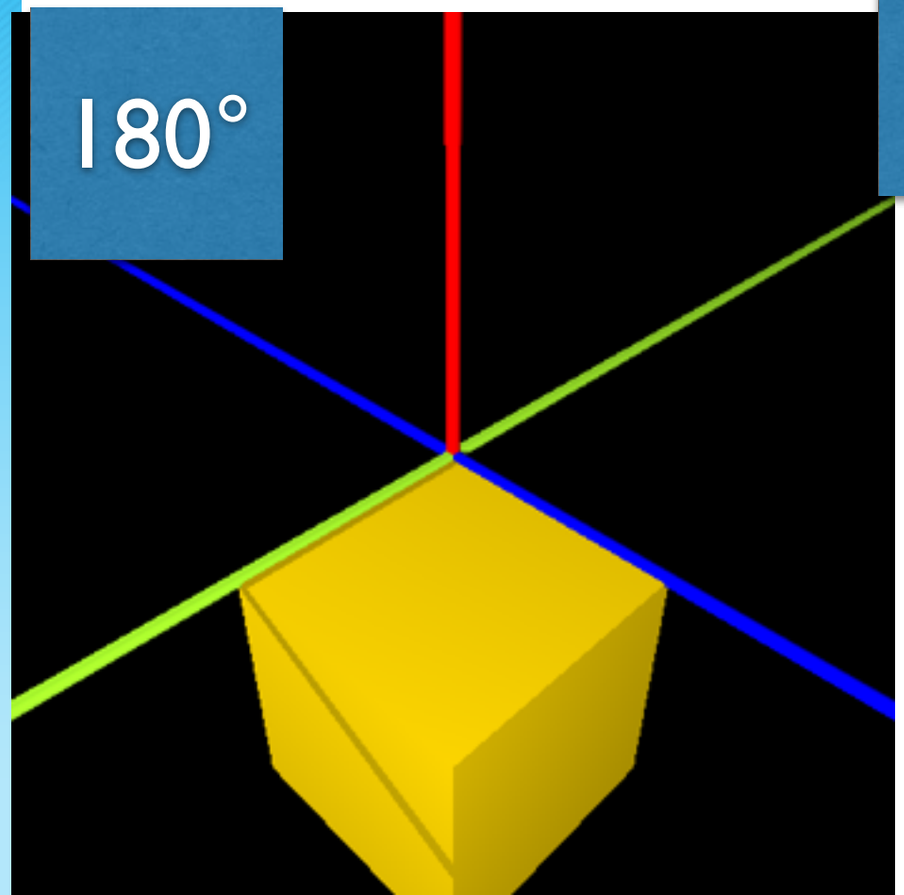
Rotation

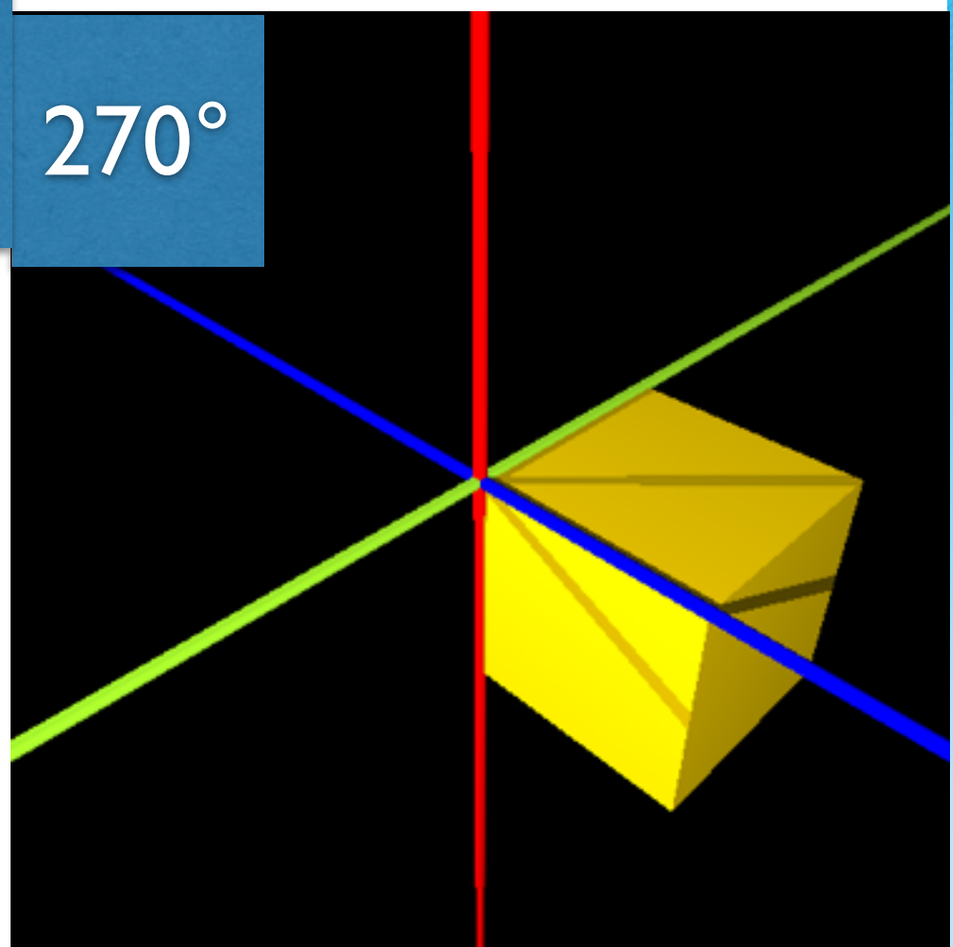
- All rotations are around their respective axis
- Rotations are counter clockwise
- Rotations are expressed in radians, not degrees

Rotation around the x-axis

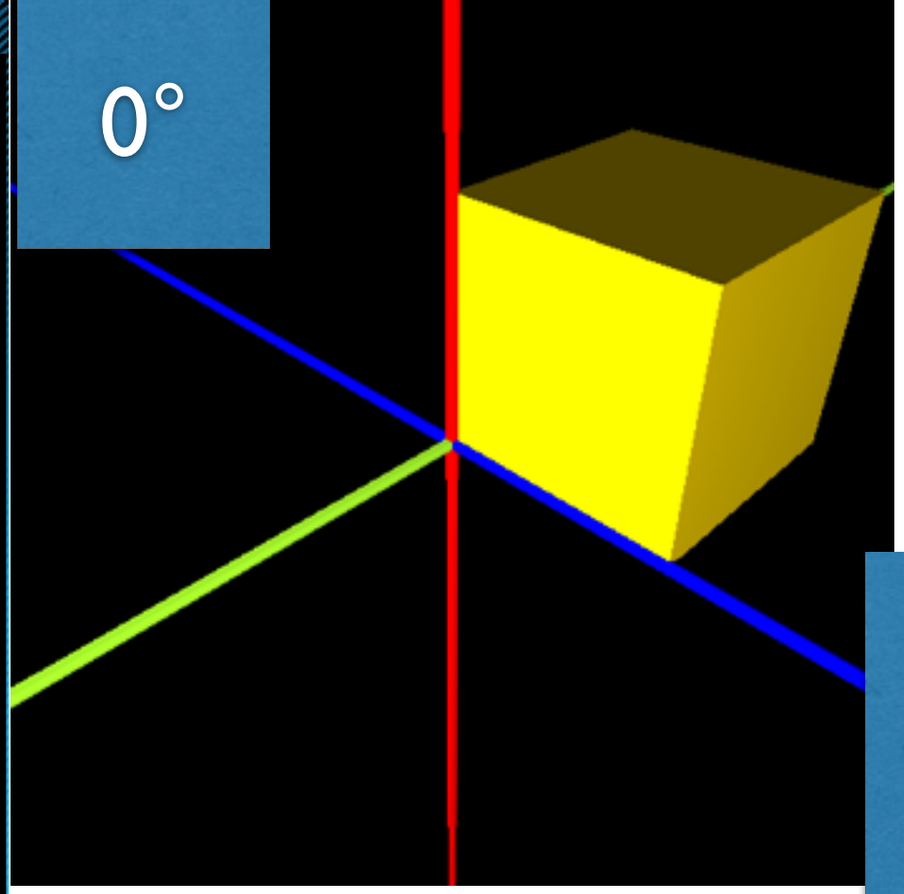


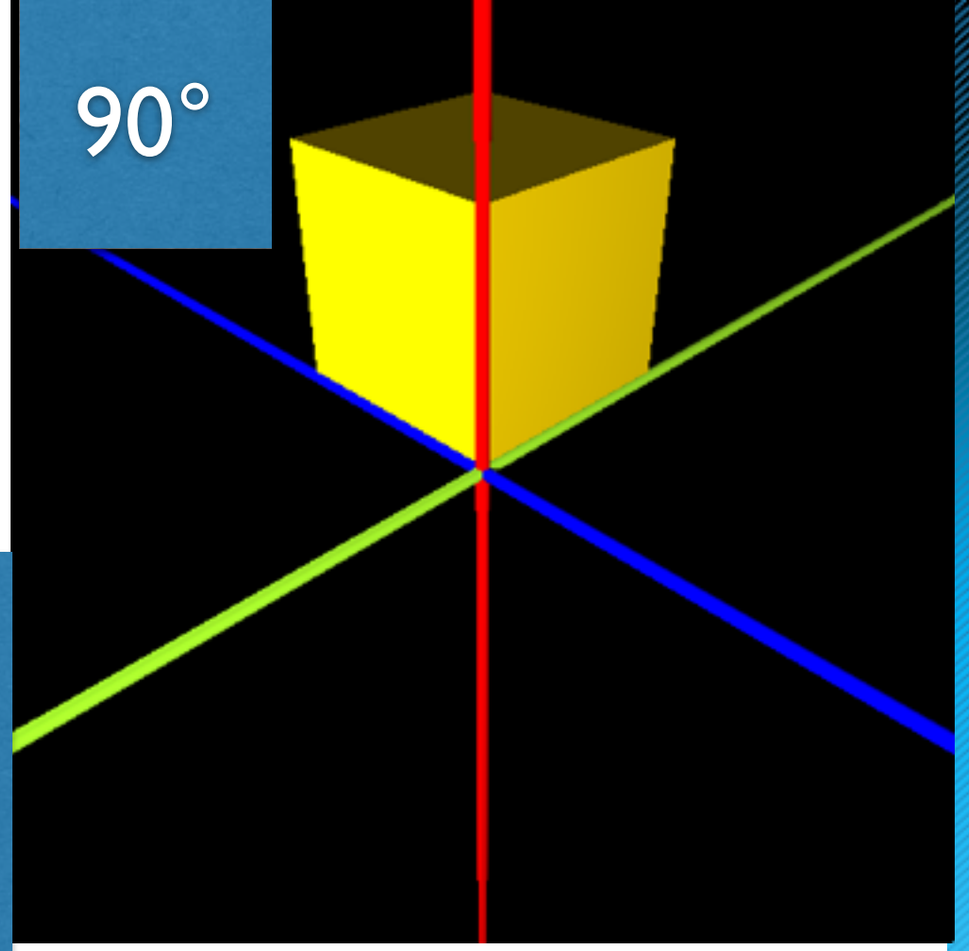


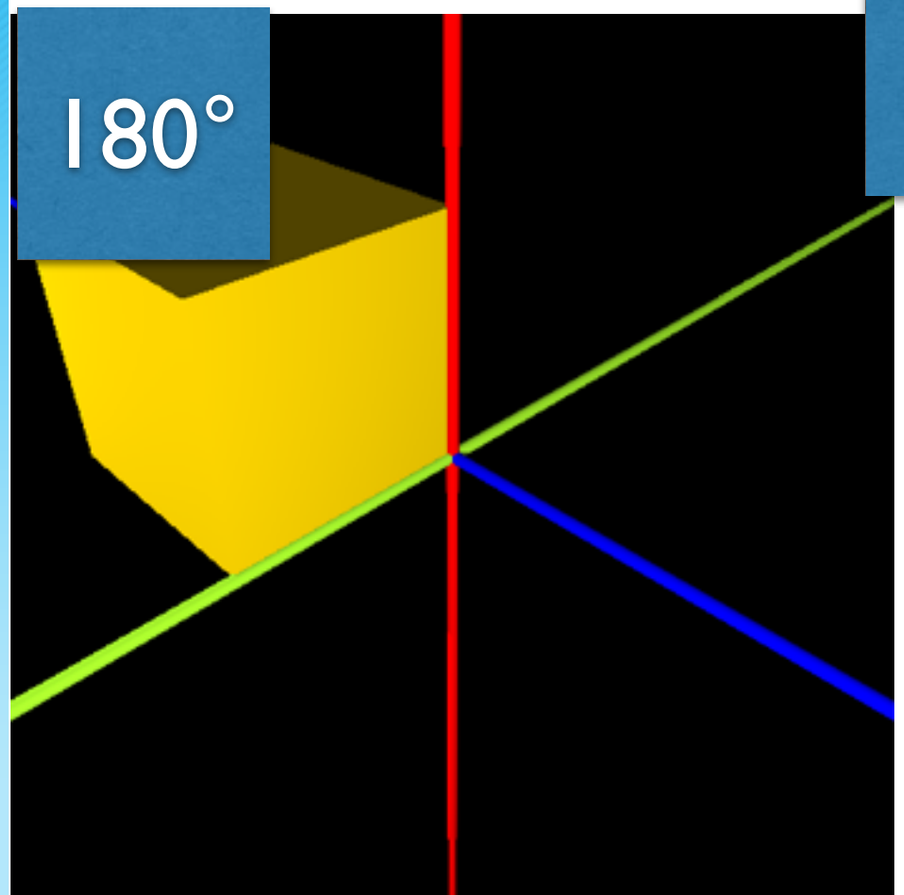


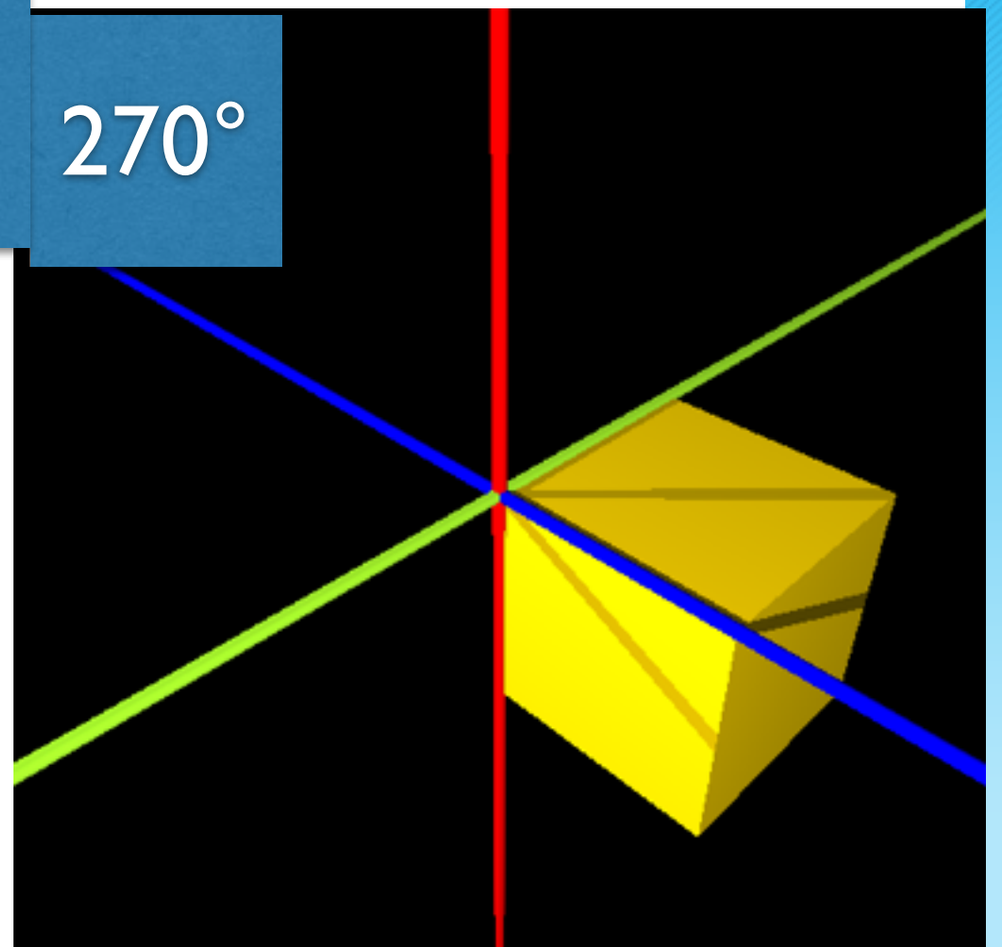


Rotation around the y-axis

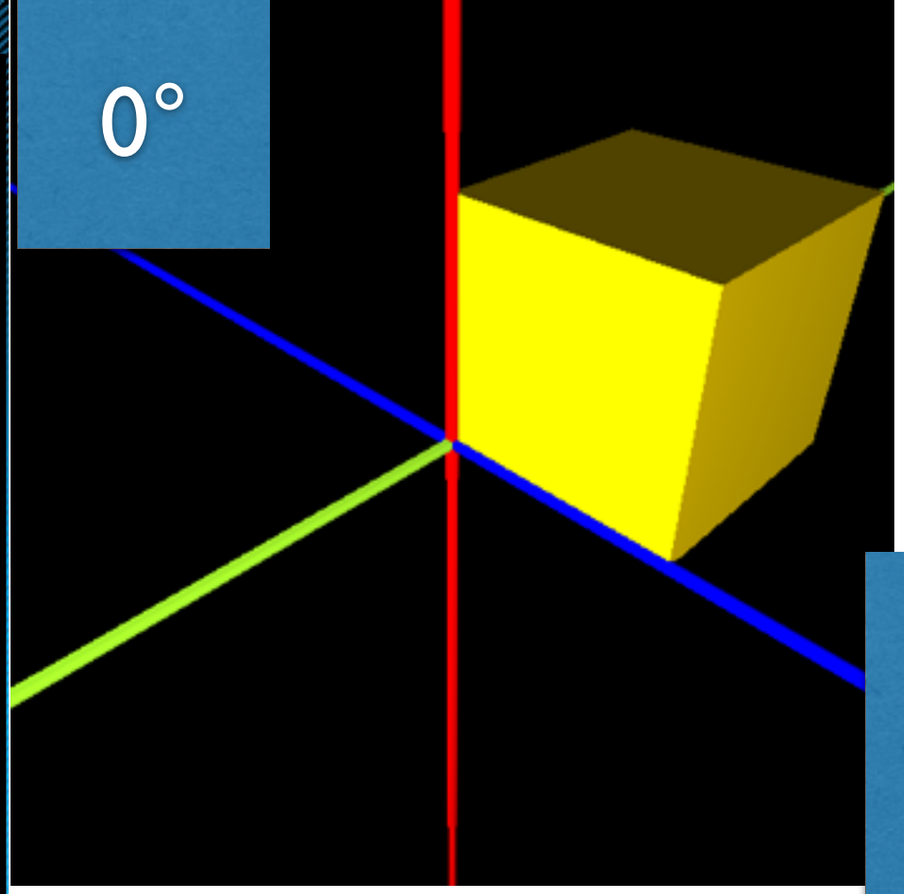


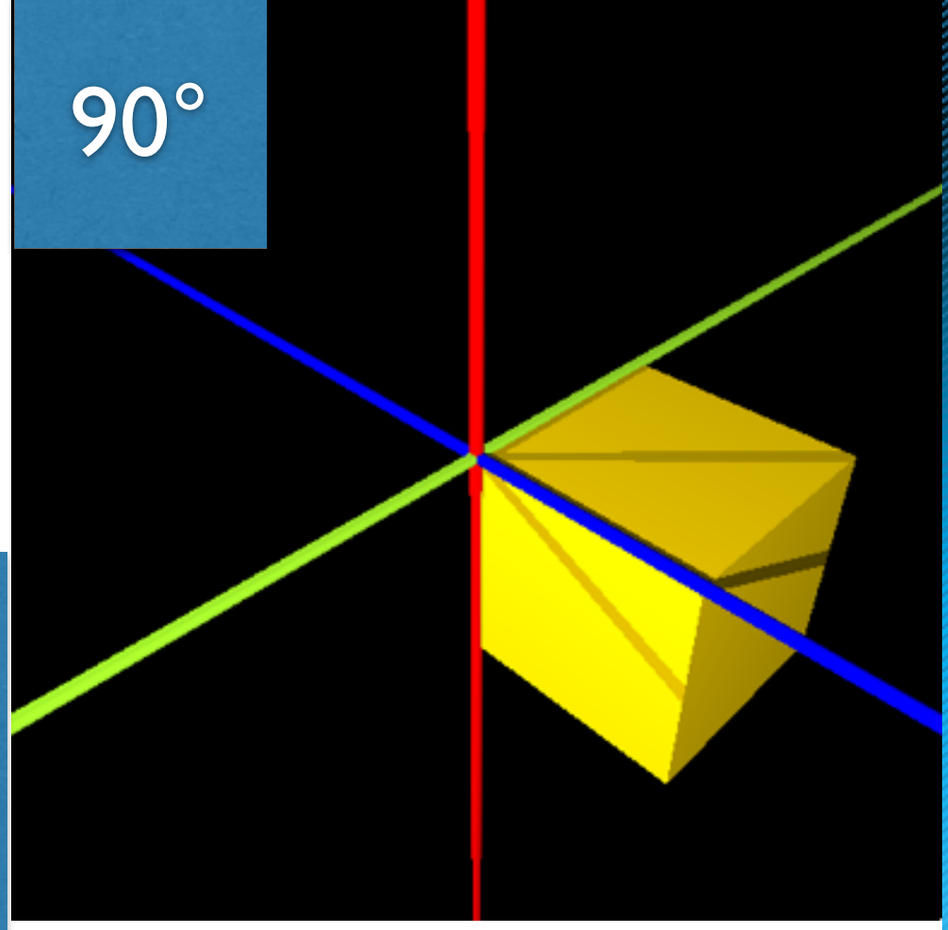


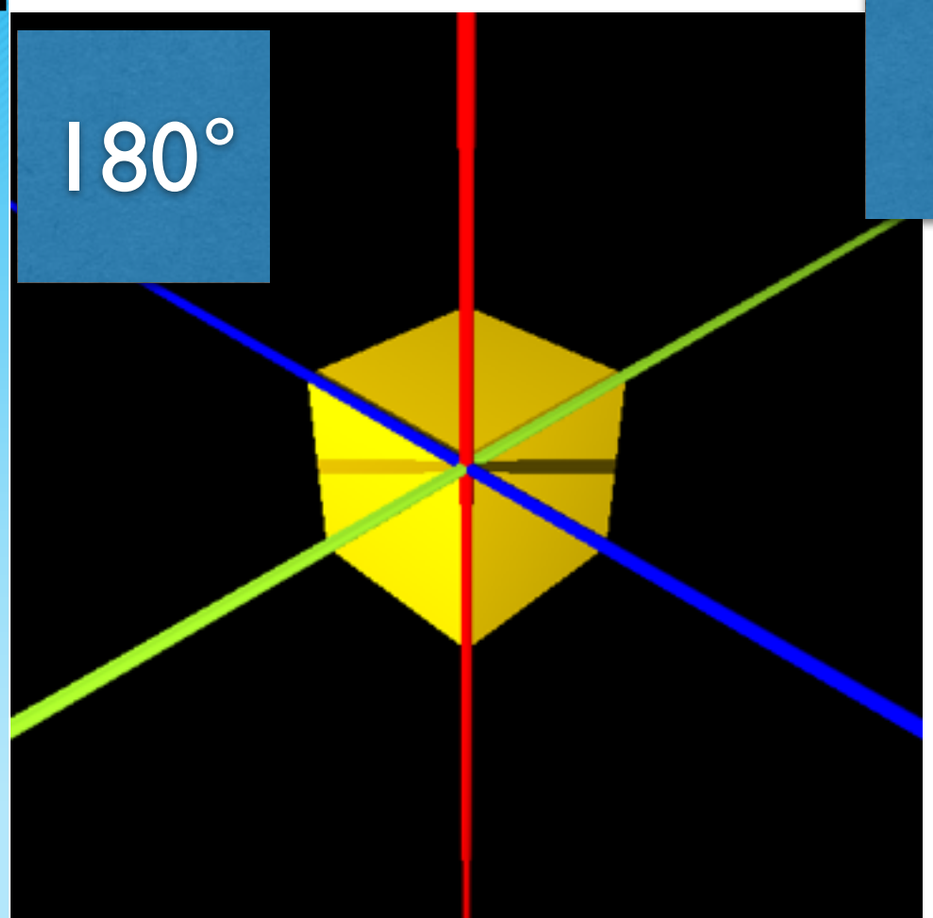


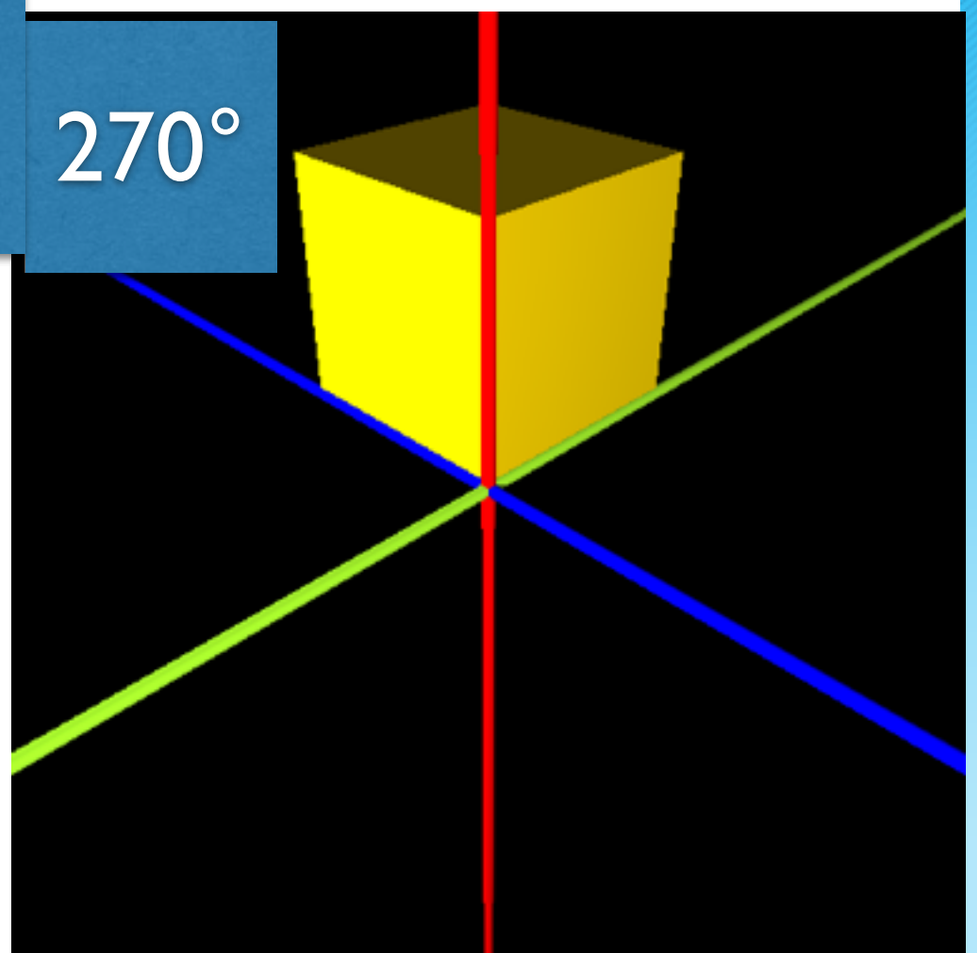


Rotation around the z-axis







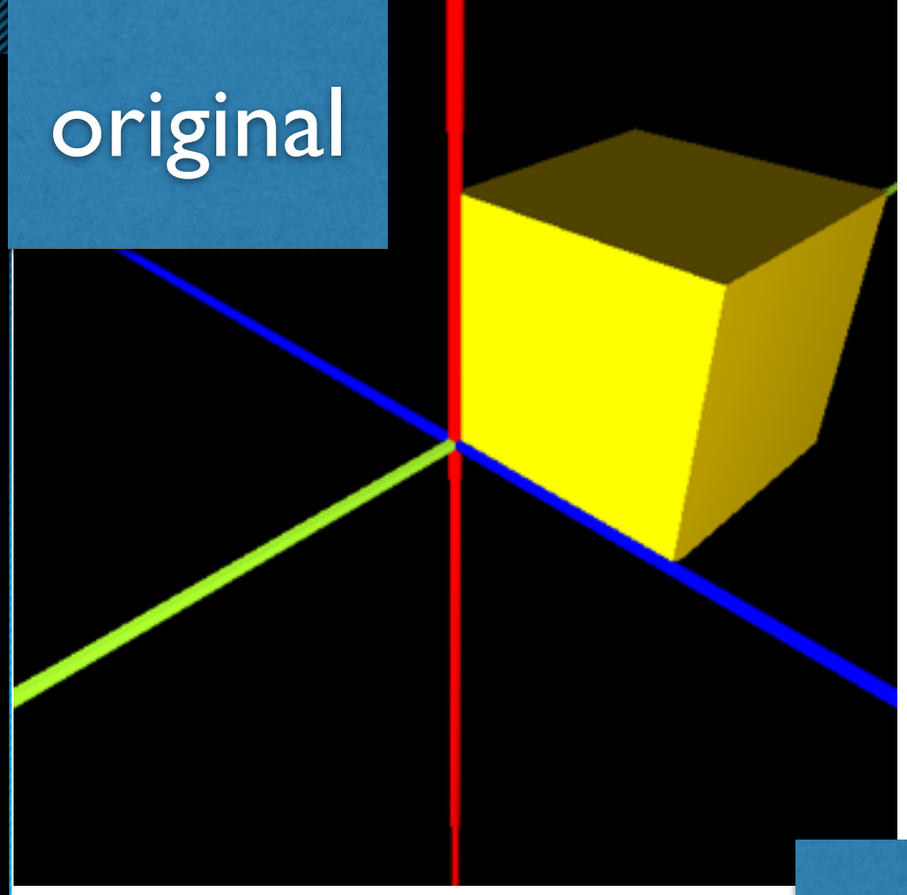


Mirroring

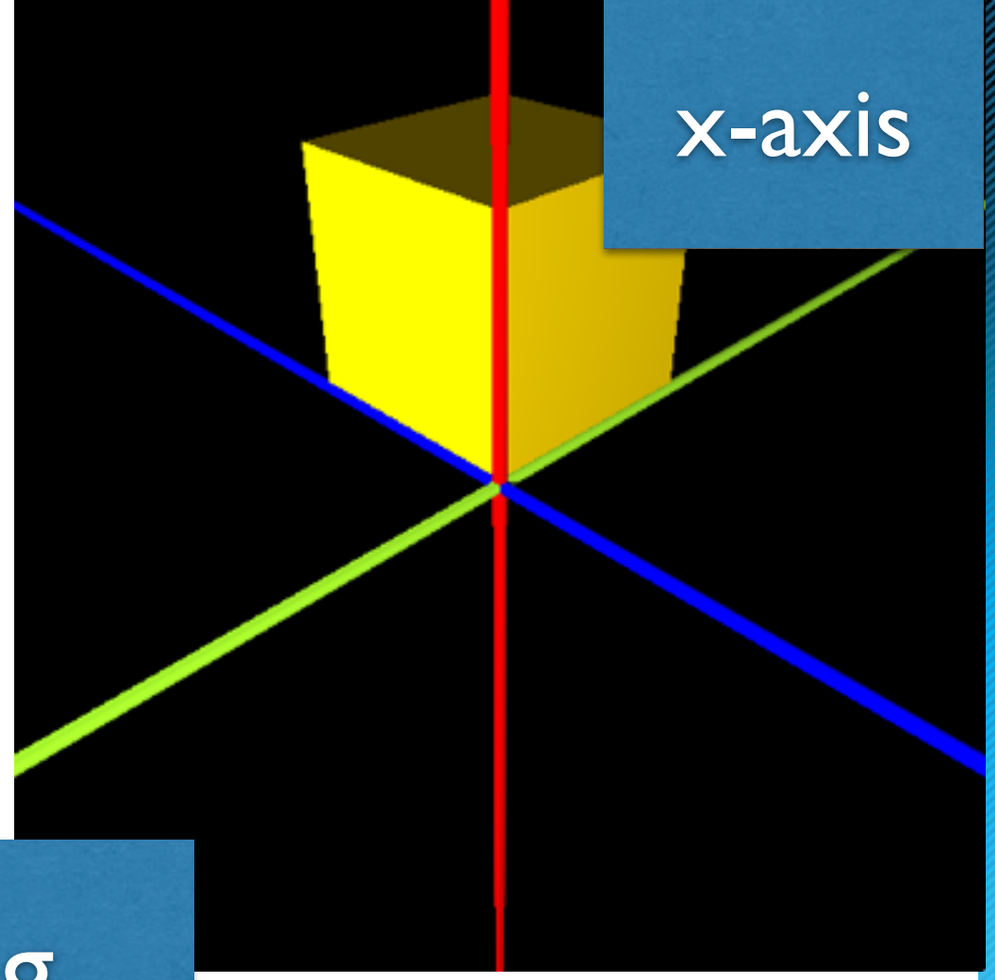
We can mirror around the different axes

Mirroring

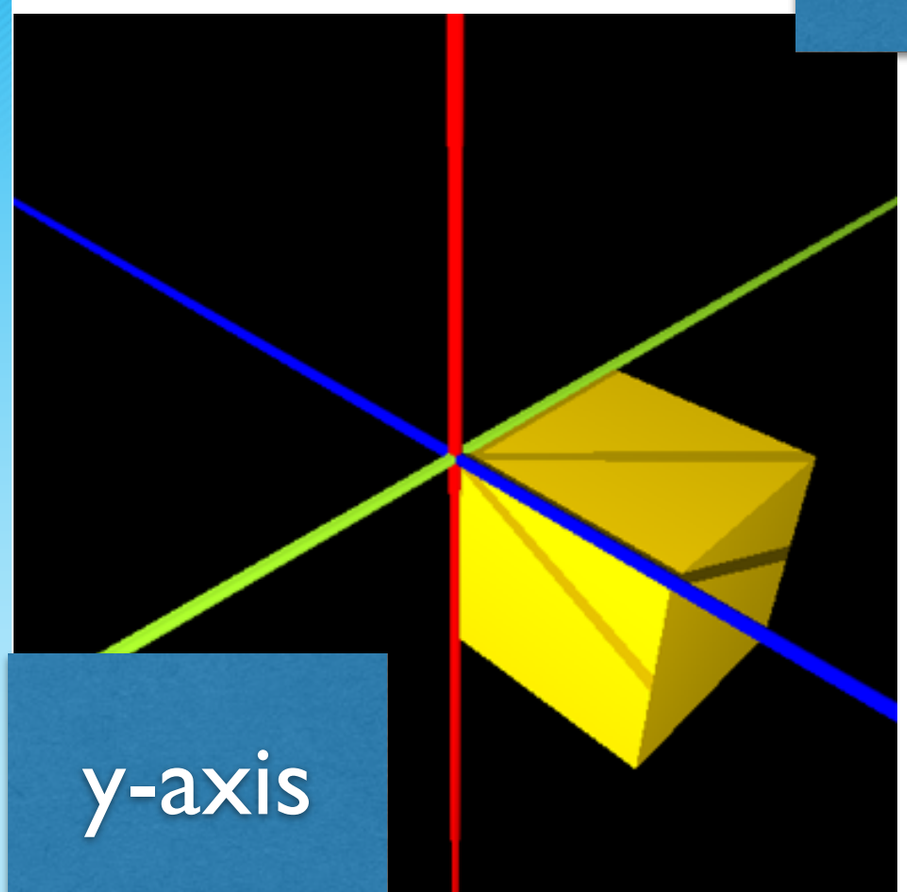
original



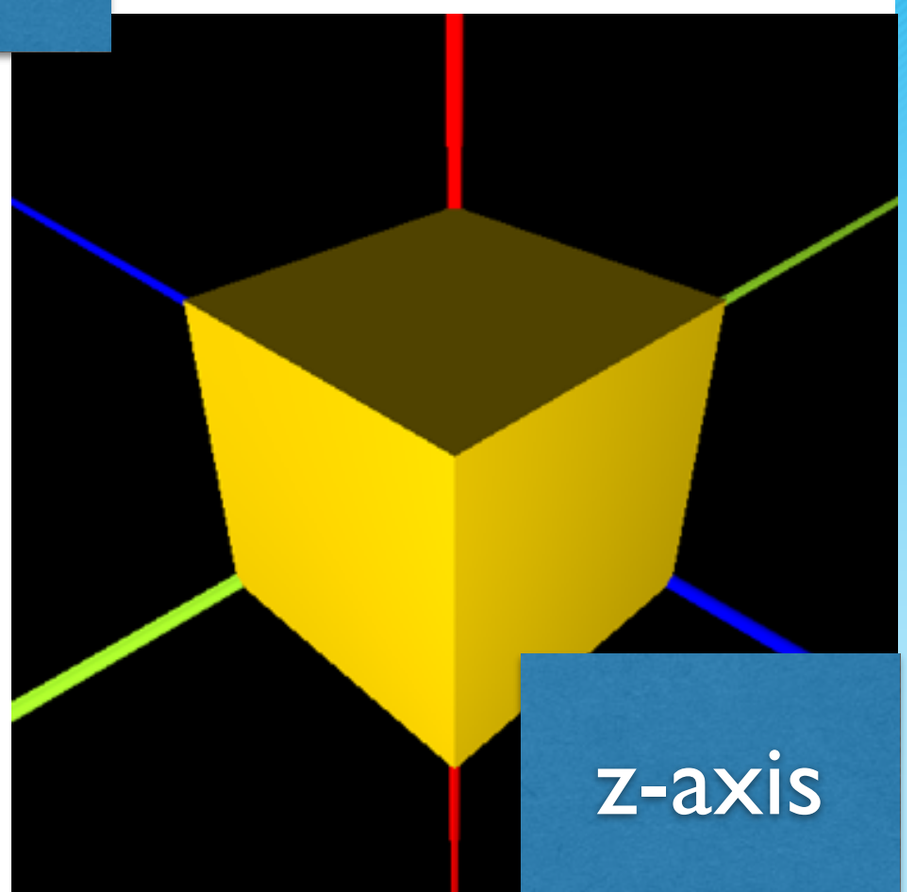
x-axis



y-axis



z-axis



Scaling

- We can scale shapes in all axes
- As with mirroring, we scale along a certain axis
- Scaling is done using floating point numbers where numbers lower than one shrinks a shape and numbers greater than one grows it (2.0 would, for instance, double the size of a shape)

Scaling by 2.0

original

x-axis

y-axis

z-axis

Shearing

- Shearing stretches a shape along an axis
- If you shear a shape along the y-axis with respect to x (written shearXY) it means that you stretch the shape along the y-axis as x increases

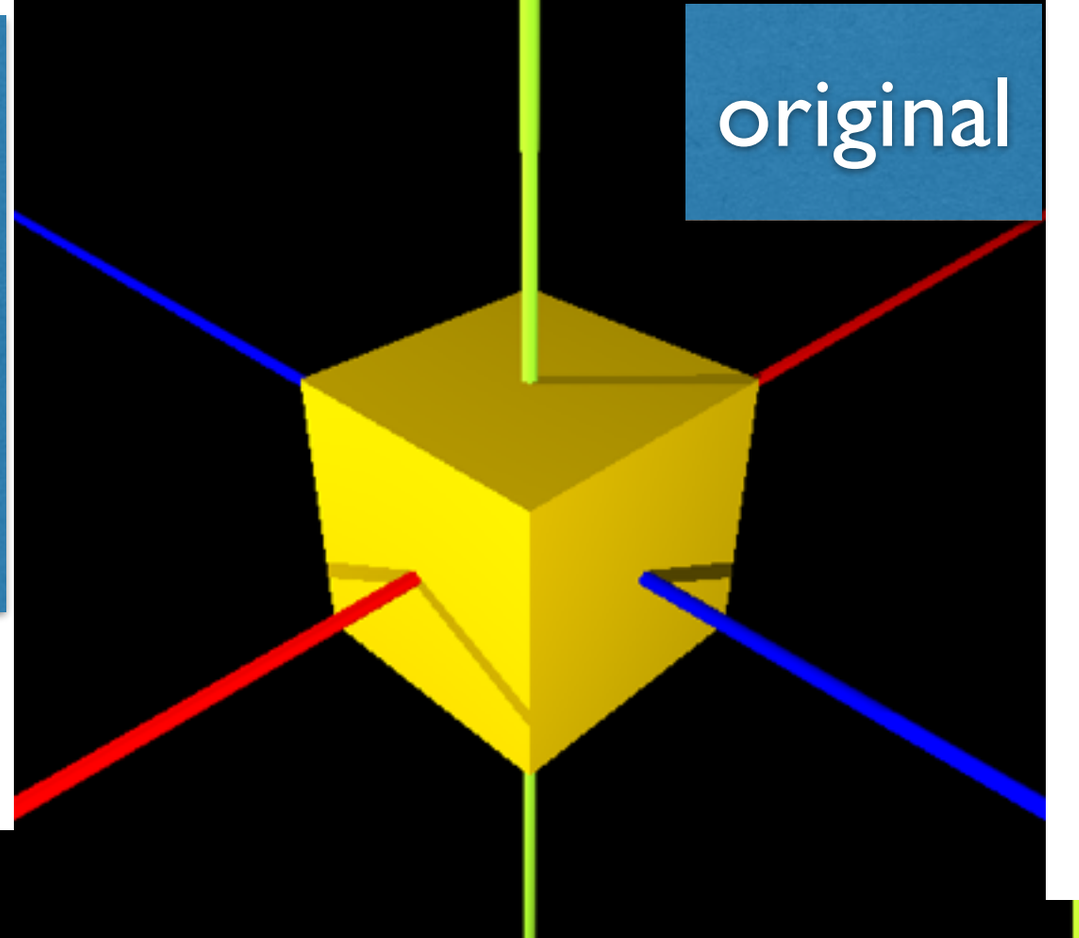
Shearing by 1.0 with respect to x

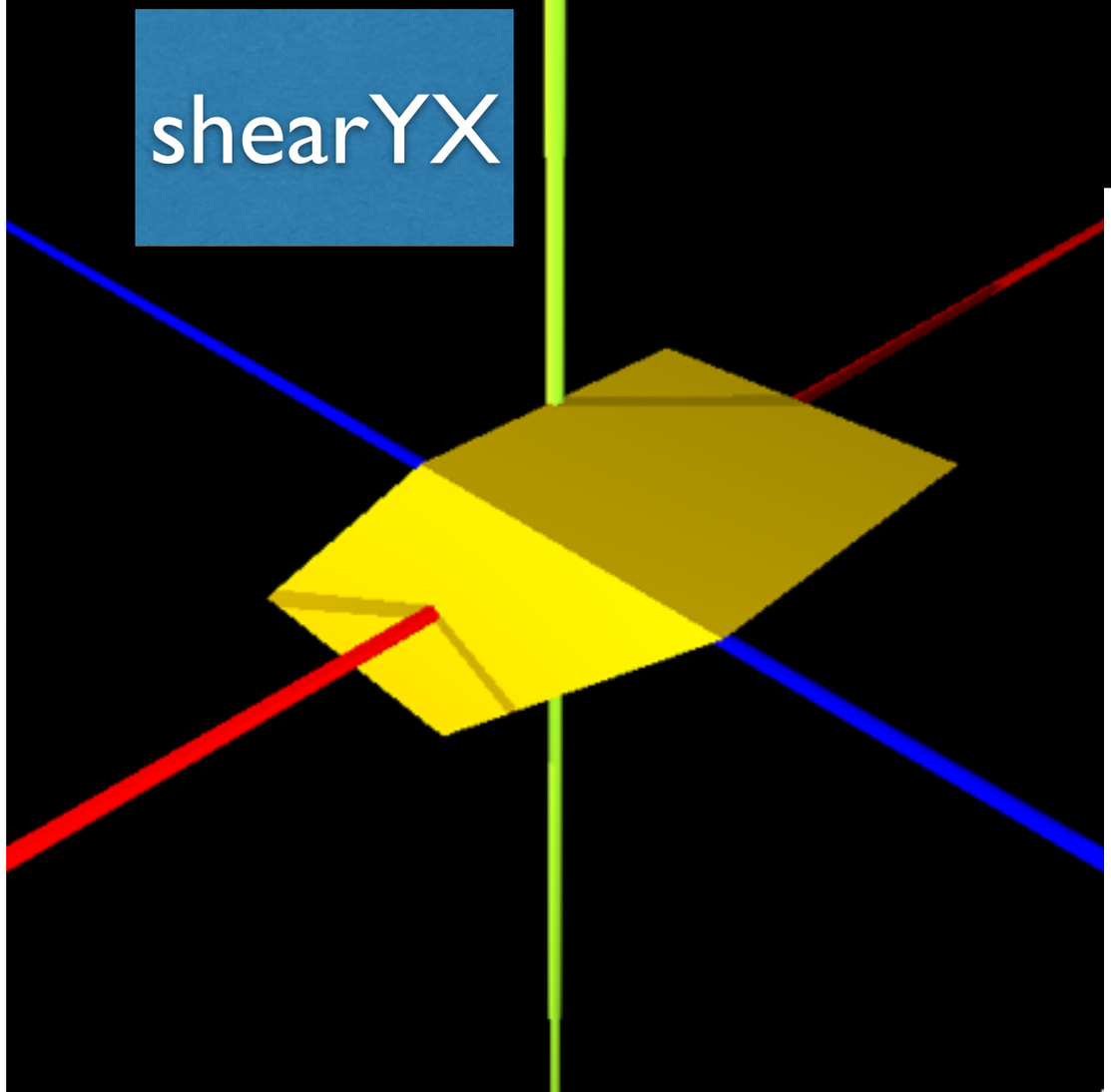
original

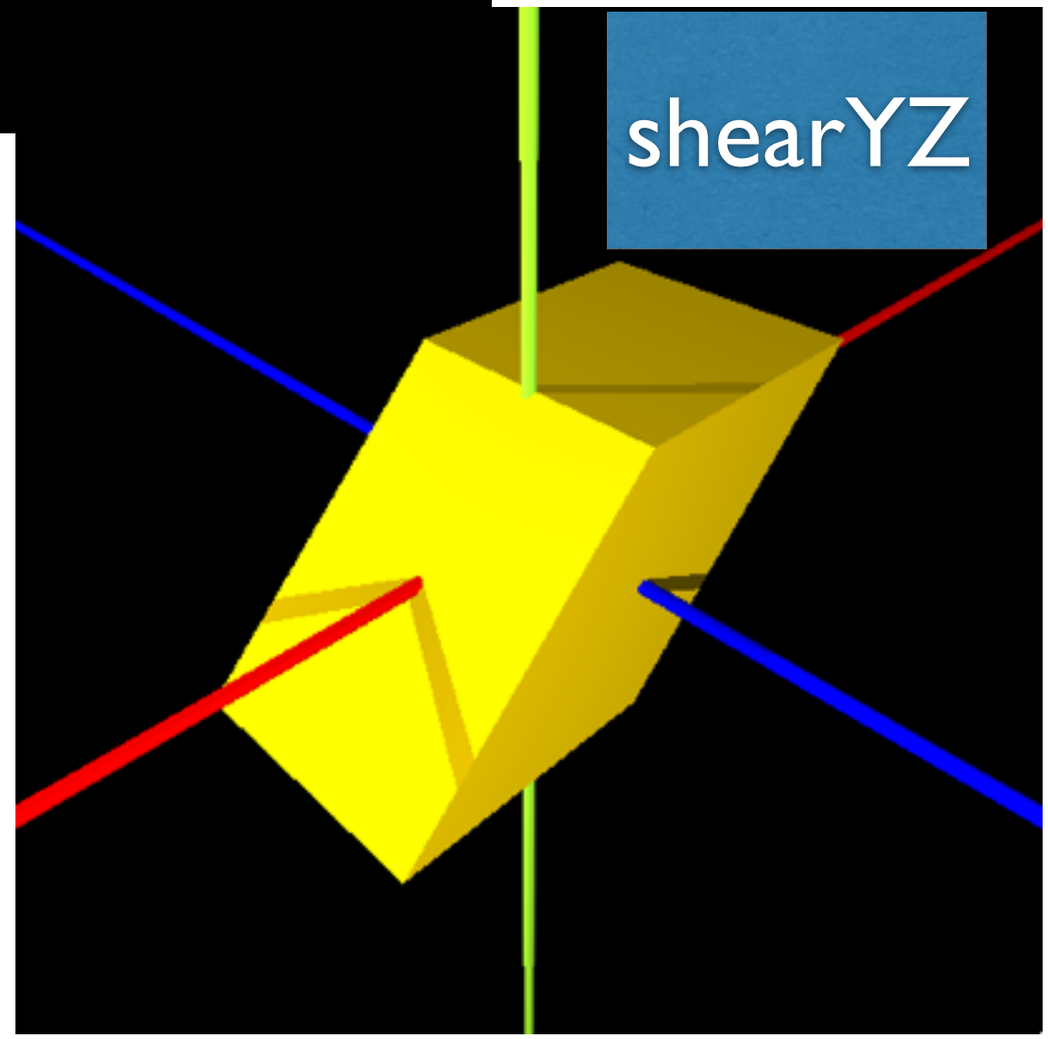
shearXY

shearXZ

Shearing by 1.0 with respect to y

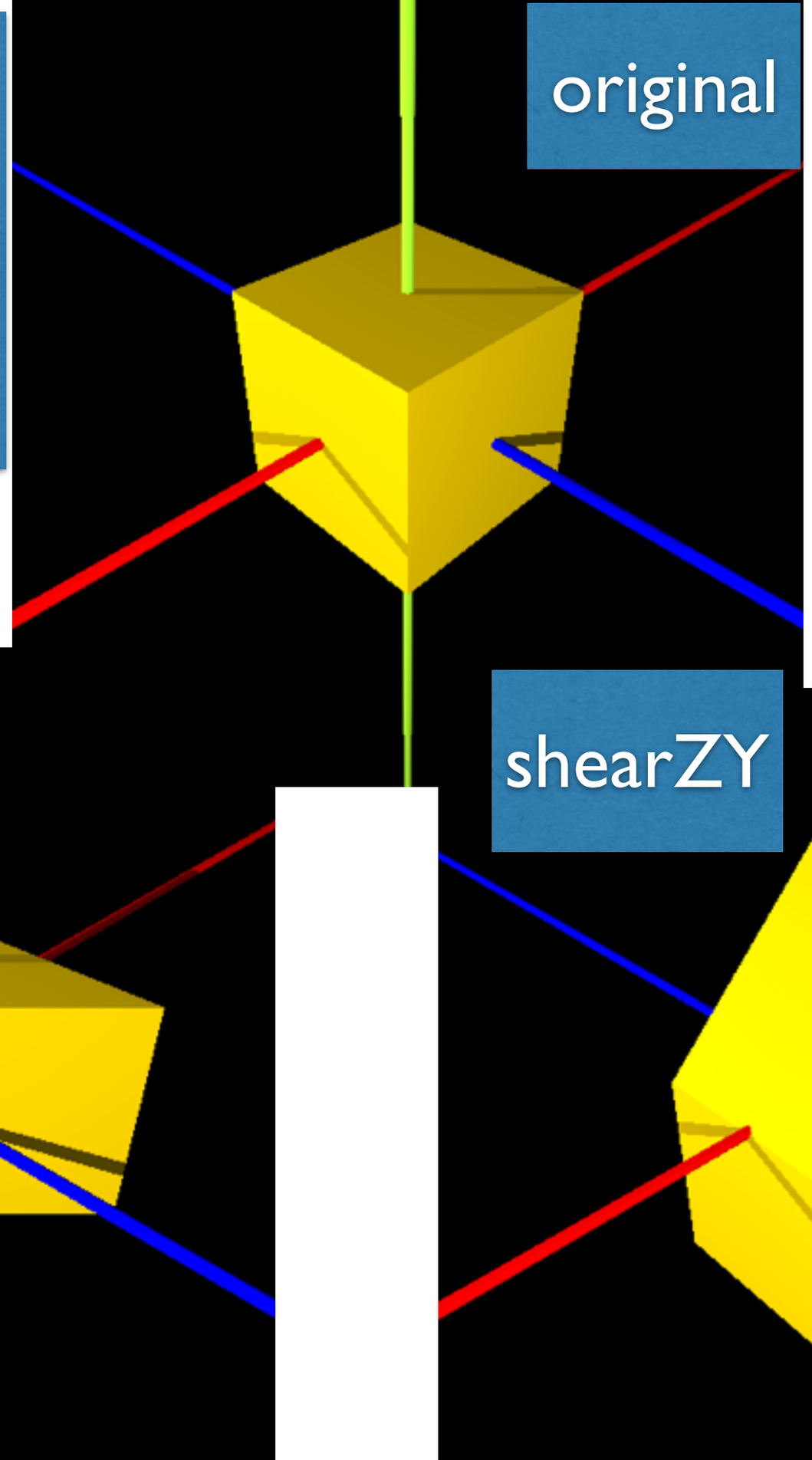




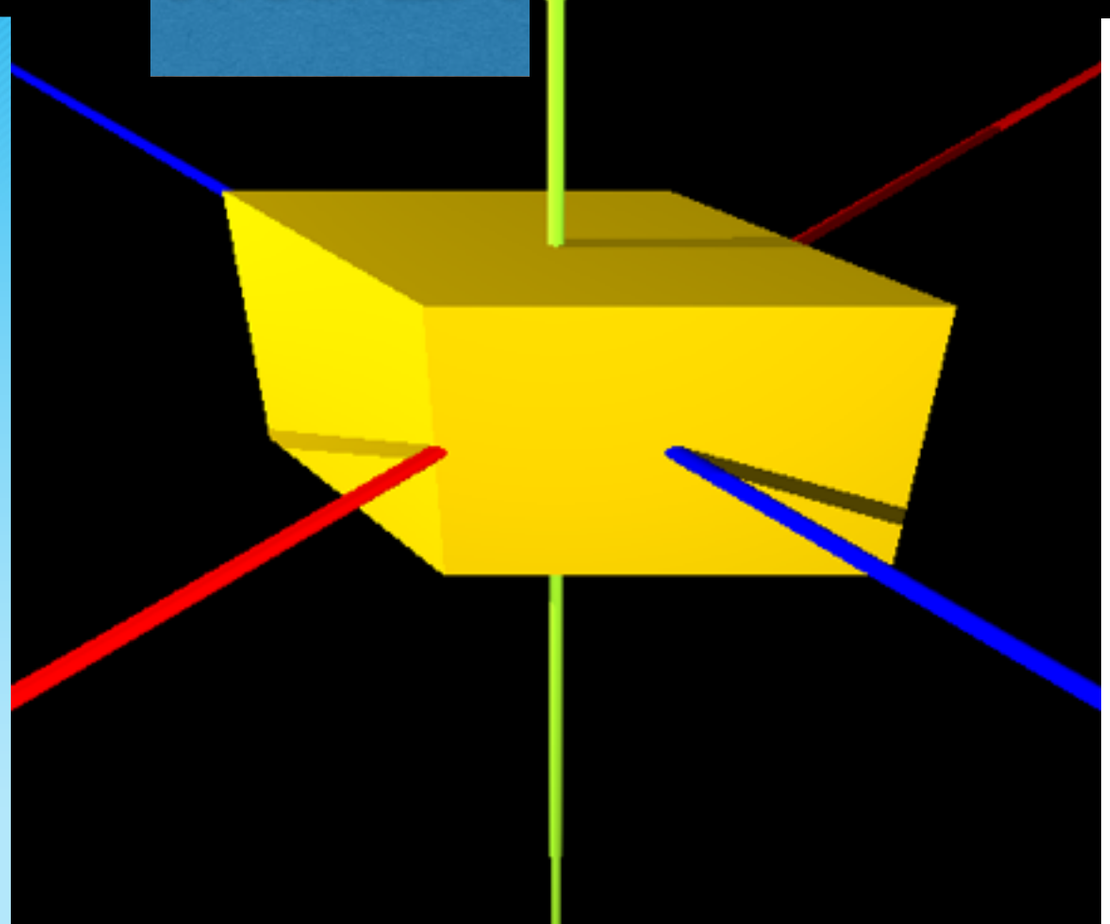


Shearing by 1.0 with respect to z

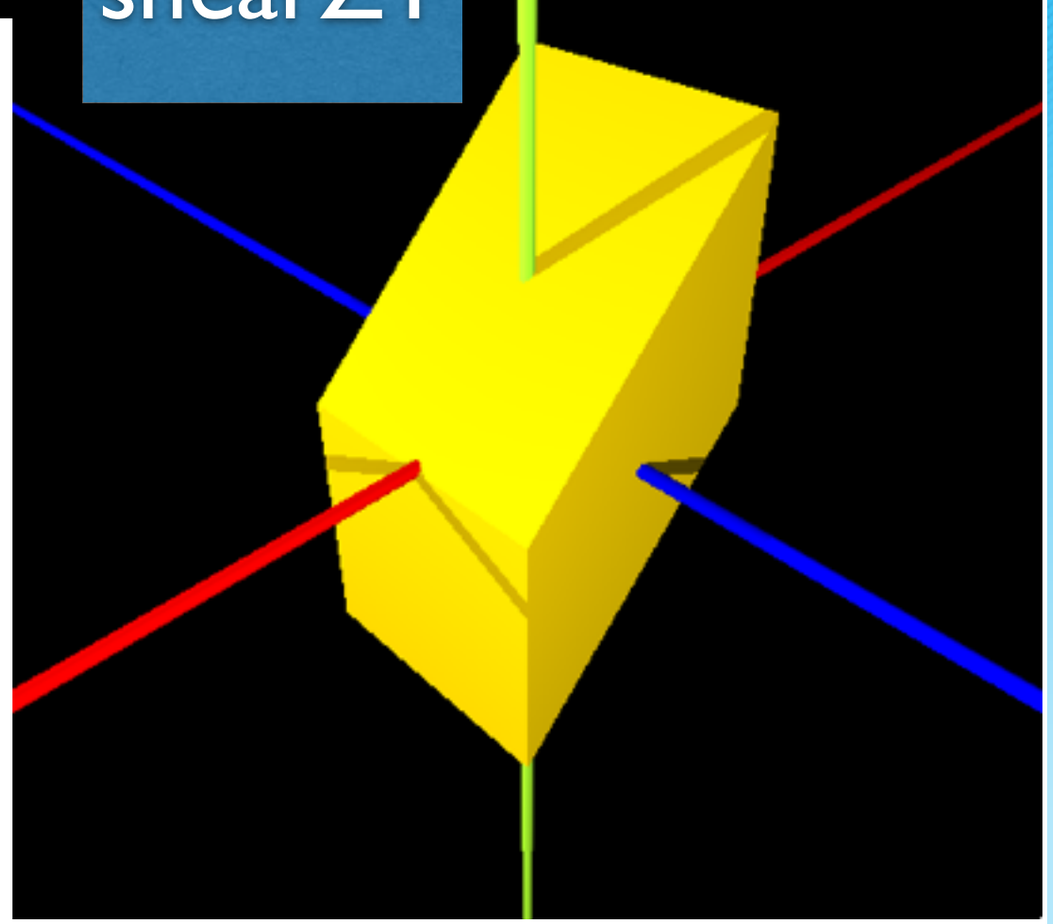
original



shearZX



shearZY



How do we do this?

- The math behind these modifications is relatively straightforward (Addition and multiplication mostly)
- The theory is a bit more hairy, but understanding it is not strictly required (but of course useful) to implement these transformations
- We use transformation matrices

Transformation matrices

- Transformation matrices are used to modify individual points and vectors (and rays by extension of that)
- In three dimensions, a transformation matrix is a **4x4** matrix (we will come to why this is in a little bit)
- To understand how these work we must first recapitulate matrix multiplication

IT UNIVERSITY OF COPENHAGEN

Matrix multiplication

Any matrix of dimension **n x m** can be multiplied by another matrix of dimension **m x p** to form a matrix of dimension **n x p**

$$\begin{pmatrix} a & b & c \\ d & e & f \end{pmatrix} * \begin{pmatrix} g & h \\ i & j \\ k & l \end{pmatrix} =$$

$$\begin{pmatrix} ag + bi + ck & ah + bj + cl \\ dg + ei + fk & dh + ej + fl \end{pmatrix}$$

Matrix multiplication

Any matrix of dimension **n x m** can be m… of dime… ix of

Our transformation matrices are all of dimension 4x4 hence the result of the multiplication will also be of that dimension

$$\begin{pmatrix} \cdots & \cdots \\ \cdots & \cdots \end{pmatrix} = \begin{pmatrix} ag + bi + ck & ah + bj + cl \\ dg + ei + fk & dh + ej + fl \end{pmatrix}$$

Modifying points

To modify a point, we put it in a 4x1 matrix (often called a vector to confuse things)

$$\begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} xa + yb + zc + d \\ xe + yf + zg + h \\ xi + yj + zk + l \\ xm + yn + oz + p \end{pmatrix}$$

Modifying points

To modify a point, we put it in a 4x1 matrix (often called a vector to confuse things)

I will talk about why we do this later

$$\begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} xa + yb + zc + d \\ xe + yf + zg + h \\ xi + yj + zk + l \\ xm + yn + oz + p \end{pmatrix}$$

Modifying points

To modify a point, we put it in a 4x1 matrix (often called a vector to confuse things)

$$\begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} xa + yb + zc + d \\ xe + yf + zg + h \\ xi + yj + zk + l \\ xm + yn + oz + p \end{pmatrix}$$

Modifying points

To modify a point, we put it in a 4x1 matrix (often called a vector to confuse things)

Important observation!!!

After we do this multiplication, we obtain a new point

Transformation matrices

- Every single transformation that we covered earlier can be expressed as a transformation matrix
- Transformation matrices can be combined using matrix multiplication to do several transformations at once
- This type of technology is heavily used by various graphics library for high-speed graphics
- Hardware support

The identity transformation

An identity matrix is a matrix with all zeroes, except for ones in the diagonal

1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

Multiplying a compatible matrix with the identity matrix returns the same matrix

The identity transformation

An identity matrix is a matrix with all zeroes, except for ones in the diagonal

All of our transformations will be extensions of the identity matrix

Multiplying a compatible matrix with the identity matrix returns the same matrix

Translation

The translation matrix moves a point

$$\begin{pmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Translation

The translation matrix moves a point

$$\begin{pmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x1 + y0 + z0 + a \\ x0 + y1 + z0 + b \\ x0 + y0 + z1 + c \\ x0 + y0 + z0 + 1 \end{pmatrix}$$

Translation

The translation matrix moves a point

$$\begin{pmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x1 + y0 + z0 + d \\ x0 + y1 + z0 + h \\ x0 + y0 + z1 + l \\ x0 + y0 + o0 + 1 \end{pmatrix} =$$

$$\begin{pmatrix} x + a \\ y + b \\ z + c \\ 1 \end{pmatrix}$$

Translation

The translation matrix moves a point

$$\begin{pmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x + a \\ y + b \\ z + c \\ 1 \end{pmatrix}$$

Note that the resulting point is on the same form that we started with, but its coordinates have been updated

Scaling

This matrix is used to scale shapes

$$\begin{pmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Scaling

This matrix is used to scale shapes

$$\begin{pmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} xa + y0 + z0 + 0 \\ x0 + yb + z0 + 0 \\ x0 + y0 + zc + 0 \\ x0 + y0 + z0 + 1 \end{pmatrix} =$$

$$\begin{pmatrix} xa \\ yb \\ zc \\ 1 \end{pmatrix}$$

Scaling

This matrix is used to scale shapes

Note that while the translation matrix moved the point by adding to its coordinates, scaling multiplies

=

xa
yb
zc
1

Mirroring

The mirror matrix inverts the sign of a point

$$\begin{pmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

a, b and c are either 1 or -1 where the latter mirrors around the x-axis for a, the y-axis for b and the z-axis for c

Rotation

The angle θ we rotate by is measured in radians

Rotation

The angle θ we rotate by is measured in radians

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotate around the x-axis

Rotation

The angle θ we rotate by is measured in radians

$$\begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotate around the y-axis

Rotation

The angle θ we rotate by is measured in radians

$$\begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotate around the z-axis

Shearing

Recall that there were six ways that we could shear a shape

1	s_{yx}	s_{zx}	0
s_{xy}	1	s_{zy}	0
s_{xz}	s_{yz}	1	0
0	0	0	1

A non-zero value for either of the s-arguments will shear the shape in the indicated direction

Composition

- You can compose transformations just by multiplying them
- Note that the order is important (rotations around an axis depends on the distance to that axis)

Putting it all together

How do we apply these transformations to our shapes?

Putting it all together

WE

DONT!!!

Putting it all together

We transform the ray instead

Transforming rays

Recall that rays have the equation

$$o + td$$

Where **o** is the origin of the ray, **t** is the distance to the hit point and **d** is the direction vector

Transforming rays

If a ray hits an shape at the point **p**, then we have that

$$o + td = p$$

However, if the shape has been transformed by a transformation matrix **T**, then **p** must have the form

$$o + td = qT$$

where **qT** = **p** and q is a point on a non-transformed shape

Transforming rays

$o + td = qT$

We obtain q in the following way

$$o + td = qT \Leftrightarrow$$

$$T^{-}(o + td) = qTT^{-} \Leftrightarrow$$

$$oT^{-} + t(dT^{-}) = q$$

If this new transformed ray hits q then it hits the transformed object

Transforming rays

$o + td = qT$

We obtain q in the following way

$$o + td = qT \Leftrightarrow$$

$$T^{-}(o + td) = qTT^{-} \Leftrightarrow$$

$$oT^{-} + t(dT^{-}) = q$$

If th[illegible]n it

Inverse matrices :(

Inverse matrices

- The bad news
  - We need to invert all the matrices We have done
- The good news
  - We know what they are. Shearing is a bit hairy, but we only need to do it once
  - We still need the regular matrices (Bounding boxes)

Transformed hit functions

Using the hit function of a shape we derive a hit function for a transformed shape

- Create a transformed ray using inverse transformation matrices
- Check if it hits the original shape
- Compute the actual hit point and normal of the transformed shape

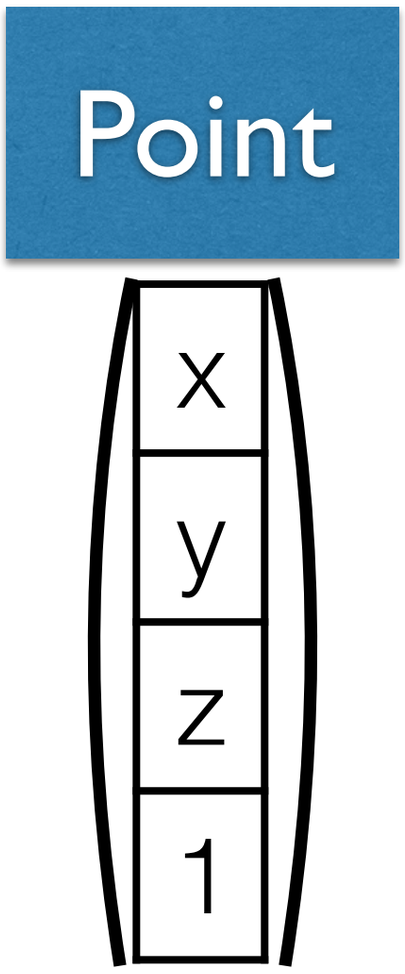
Three problems

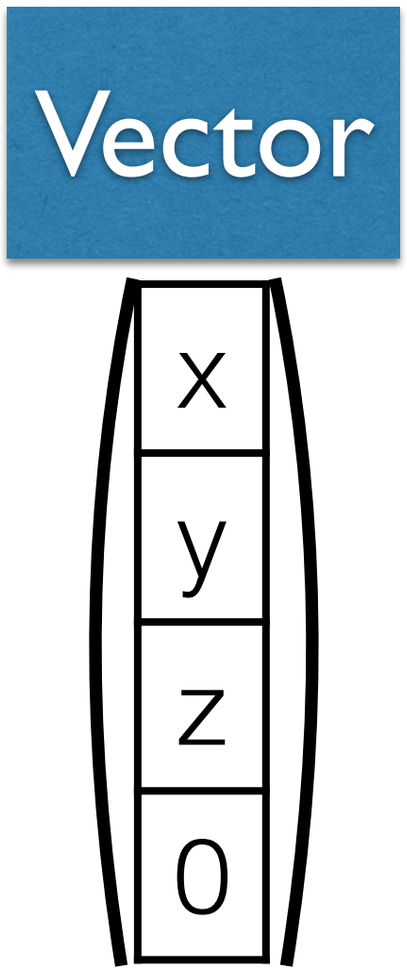
There are things we must solve

- How do we transform vectors? (we have only done points)
- If we hit a shape with a transformed ray, where does the original ray hit the transformed shape?
- What is the normal of this hit point?

Transforming normals

Both points and vectors are stored in one-dimensional matrices (vectors) but with one key difference





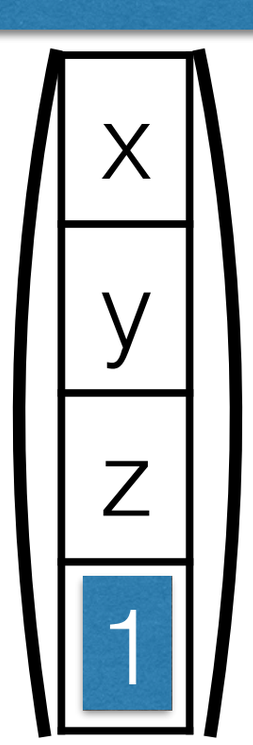
Transforming normals

Both points and vectors are stored in one-dimensional matrices (vectors) but with one key difference

Transforming normals

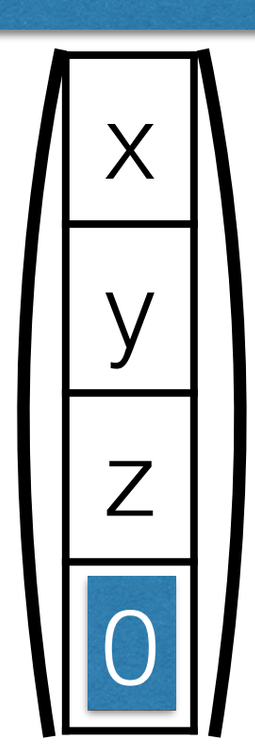
Both points and vectors are stored in one-dimensional matrices (vectors) but with one key difference

Point



We never translate vectors!

Vector



Hit point

If we hit a shape with a transformed ray, where does the original ray hit the transformed shape?

Hit point

If we hit a shape with a transformed ray, where does the original ray hit the transformed shape?

The distance is exactly the same — plug the distance into the non-transformed ray equation and you have your hit point

Hit point normal

What is the normal of the transformed hit point?

Transform the normal using the inverse transposed transformation matrix

Hit point normal

What is the normal of the transformed hit point?

Transform the normal using the inverse transposed transformation matrix

Transposed matrices

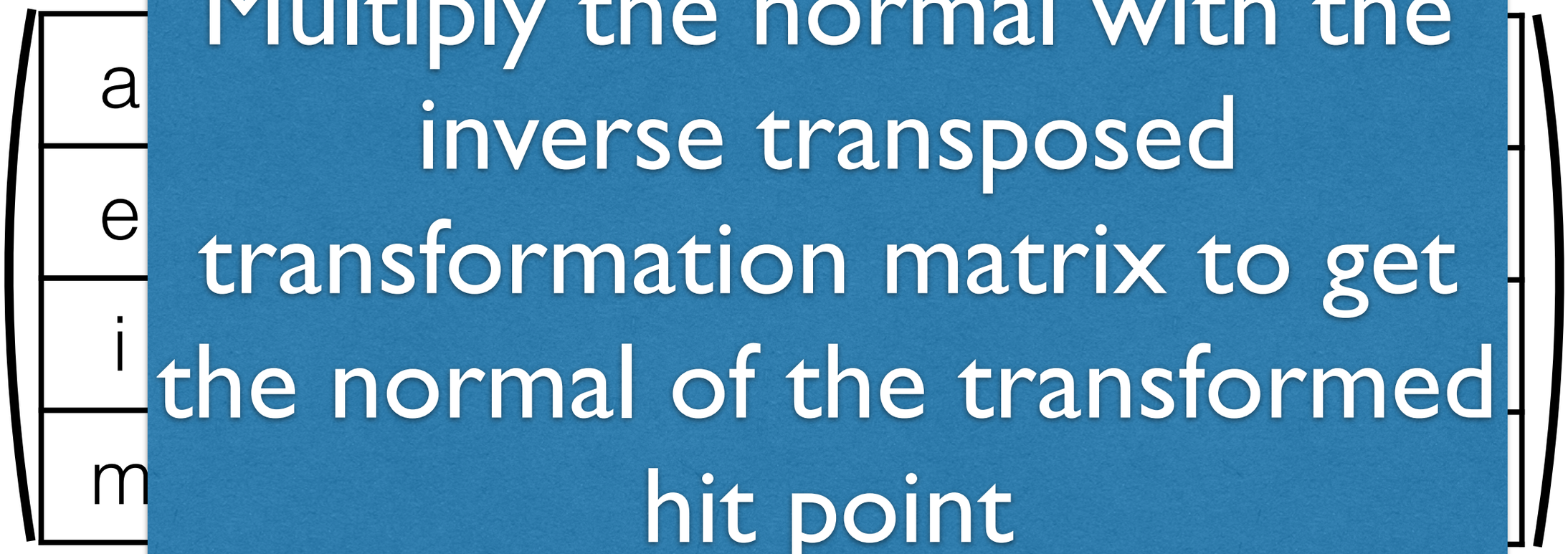
The transpose of a matrix swaps all rows with columns

$$\begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{pmatrix}^T = \begin{pmatrix} a & e & i & m \\ b & f & j & n \\ c & g & k & o \\ d & h & l & p \end{pmatrix}$$

Note that the diagonal is the same

Transposed matrices

The transpose of a matrix swaps all



Multiply the normal with the inverse transposed transformation matrix to get the normal of the transformed hit point

Note that the diagonal is the same

So what are the inverses?

The only non-trivial inverse is shearing

Translation inverse

Normal

$$\begin{pmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Inverse

$$\begin{pmatrix} 1 & 0 & 0 & -a \\ 0 & 1 & 0 & -b \\ 0 & 0 & 1 & -c \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Scaling inverse

Normal

$$\begin{pmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Inverse

$$\begin{pmatrix} 1/a & 0 & 0 & 0 \\ 0 & 1/b & 0 & 0 \\ 0 & 0 & 1/c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Mirroring inverse

Normal

$$\begin{pmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Inverse

$$\begin{pmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The mirror matrix is its own inverse
(remember that a must be 1 or -1)

Rotation inverse (x-axis)

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Normal

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Inverse

Rotation inverse (y-axis)

$$\begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Normal

$$\begin{pmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Inverse

Rotation inverse (z-axis)

$$\begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Normal

$$\begin{pmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Inverse

Shearing

$$\begin{pmatrix} 1 & s_{yx} & s_{zx} & 0 \\ s_{xy} & 1 & s_{zy} & 0 \\ s_{xz} & s_{yz} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Just copy this

$$\begin{pmatrix} 1-s_{yz}s_{zy} & -s_{yx}+s_{yz}s_{zx} & -s_{zx}+s_{yx}s_{zy} & 0 \\ -s_{xy}+s_{xz}s_{zy} & 1-s_{xz}s_{zx} & -s_{zy}+s_{xy}s_{zx} & 0 \\ -s_{xz}+s_{xy}s_{yz} & -s_{yz}+s_{xz}s_{yx} & 1-s_{xy}s_{yx} & 0 \\ 0 & 0 & 0 & D \end{pmatrix} D^{-1}$$

$$D = 1 - s_{xy}s_{yx} - s_{xz}s_{zx} - s_{yz}s_{zy} + s_{xy}s_{yz}s_{zx} + s_{xz}s_{yx}s_{zy}$$

IT UNIVERSITY OF COPENHAGEN

Questions?

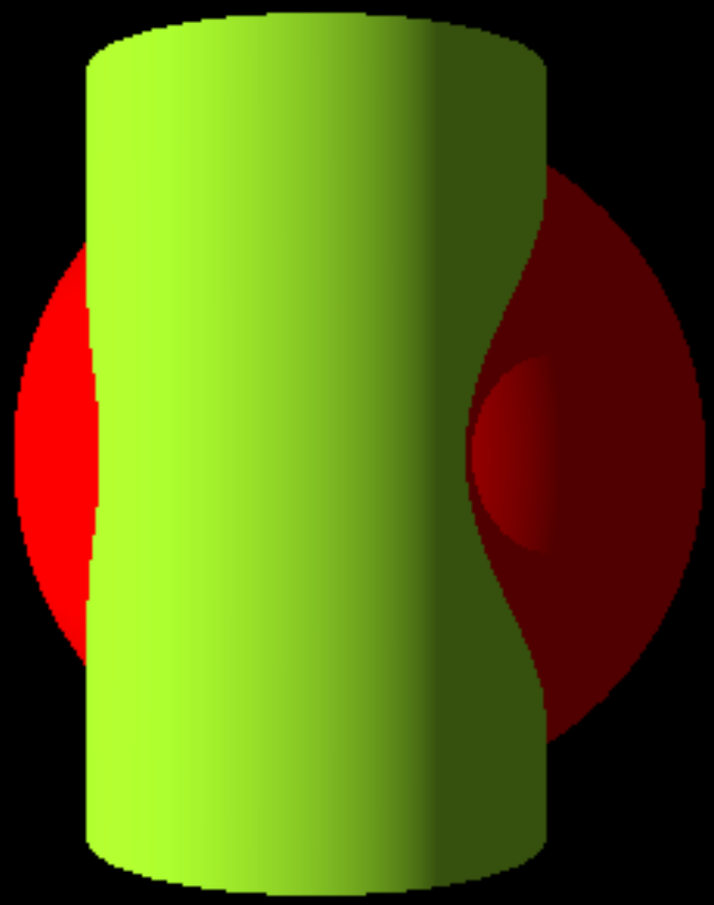
Constructive Solid Geometry

CSG

- Constructive Solid Geometry is a means to construct complex shapes out of simple ones (spheres, cubes, cylinders, et.c.
- The shapes must be solid (tori are solid, but planes or open hemispheres are not
- Simple language consists of union, intersection and subtraction

IT UNIVERSITY OF COPENHAGEN

Union



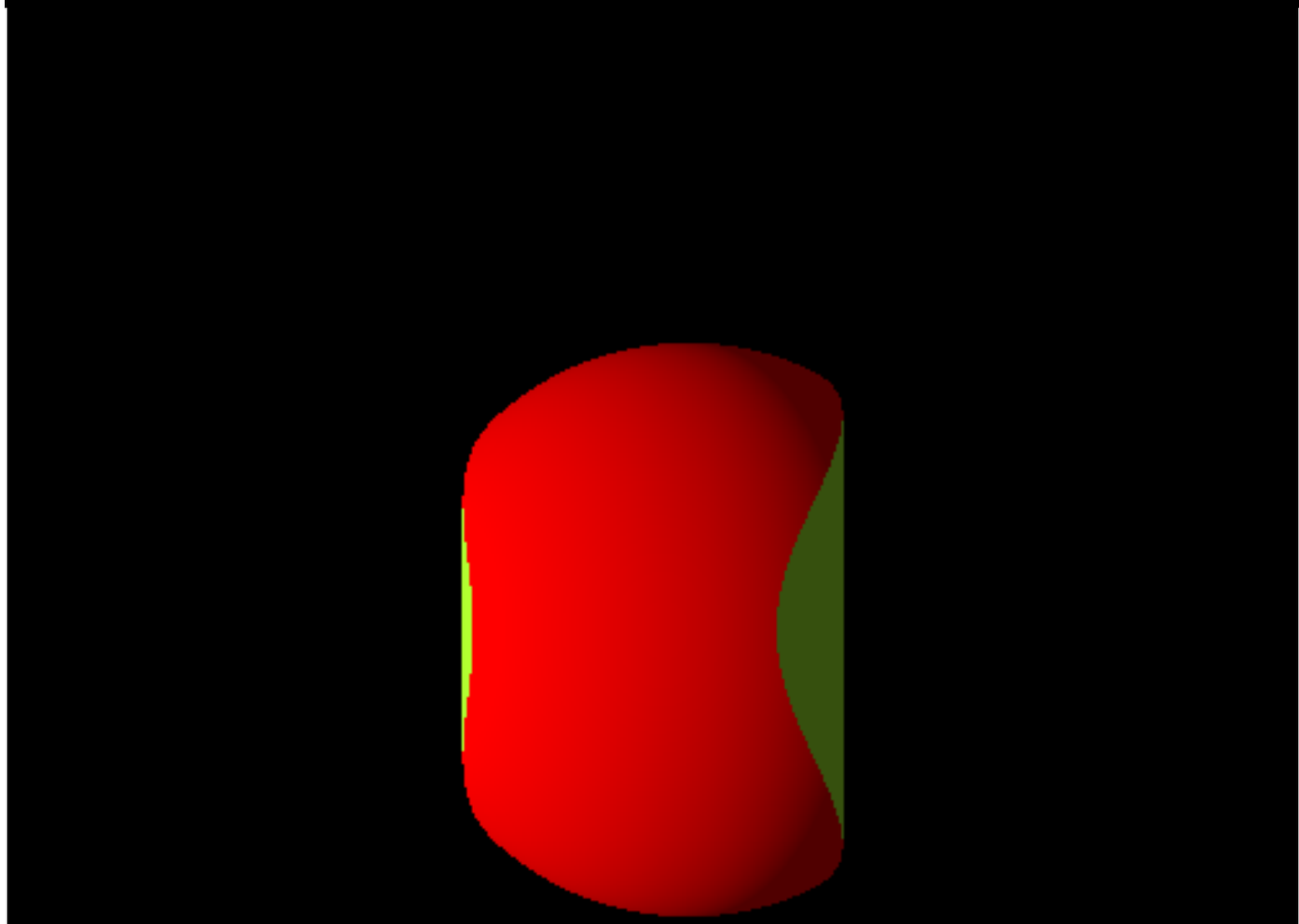
Unions compose two shapes

Second Year Project 2016

Jesper Bengtson

ITU UNIVERSITY OF COPENHAGEN

Intersection



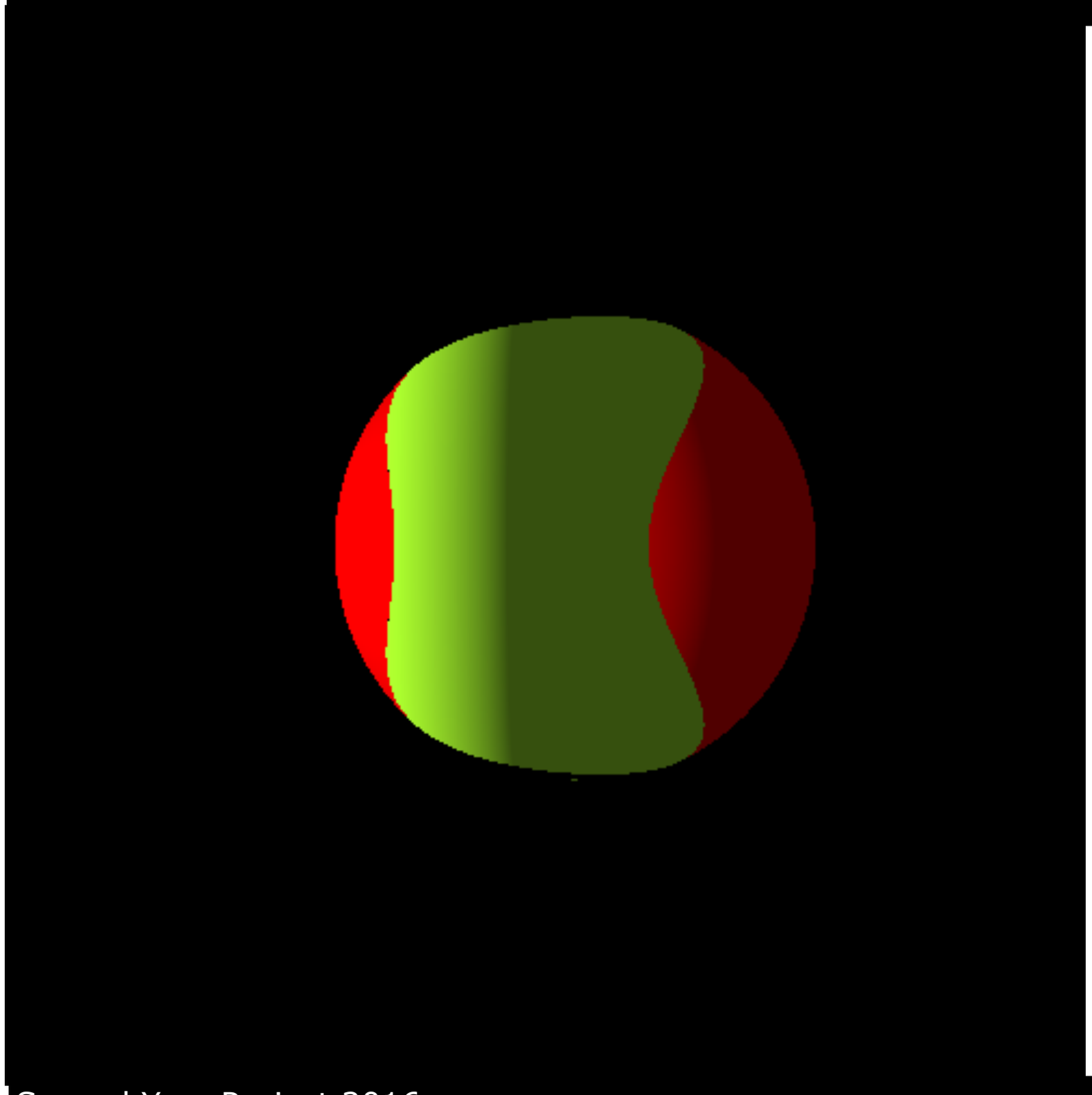
Intersection keeps the difference between two shapes

Second Year Project 2016

Jesper Bengtson

ITU UNIVERSITY OF COPENHAGEN

Subtraction



Subtraction subtracts one shape from another

Second Year Project 2016

Jesper Bengtson

Assumptions

- You may assume that there are now internal edges within the shapes
- However, you must ensure that after your operations there still are no internal edges within your shape (union for instance)

Assumptions

- Yo
  int
- Ho
  yo
  int
  (u

Tip!

When you create your new hit function for new shapes you can ignore hits to surfaces within a shape and look for other ones

Shape interface

- You are strongly encouraged to create a function that determines wether or not you are inside another shape or not
- This function must only be available for solid shapes
- Your functions are allowed to fail if the user tries to compose open shapes

IT UNIVERSITY OF COPENHAGEN

Questions?