Computer Graphics

Discussion 4

Garett Ridge and Sam Amin
garett@cs.ucla.edu samamin@ucla.edu

Advanced Project 2 Topics

- Scene Graphs – manipulating a scene tree & its transforms
- Procedural textures (perlin noise)
- Procedural shapes (plants) – look up L-systems
- Shadows (Projection onto surfaces)
- Advanced shading: caustics & reflection maps, bump mapping, displacement maps
- Collision detection (shape intersection tests)
- Linear / rotational momentum w/ sudden impulses

Advanced Project 2 Topics

- Shader particle effects (sparkles, fire, smoke, splashing, etc.)
- Spring/damper physics
- Curved interpolated shapes & movement interpolation - “tweening”
- Camera interpolation - Quaternions
- Import OBJ files (new shapes from files)
- Marching cubes / drawing implicit fields
- Fractal generation & rendering – look up Mandelbulb videos
- Mouse picking

Advanced Project 2 Topics

- Telling a story about the objects you manage to make
- Making an advanced topic interactive / user controllable to create a challenge

Part IV: Drawing Practice for Midterm

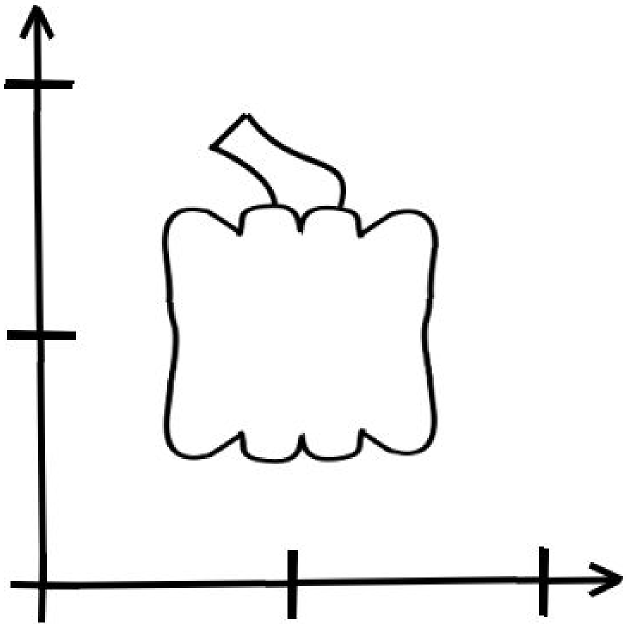
shapes.js, Shapes.cpp

Throwback--
Matrix Mult is Not Commutative

Repeat after me

Drawing example: Pumpkin (Pretend it's fall quarter)

Given this pumpkin at (1,1),



Drawing example: Pumpkin

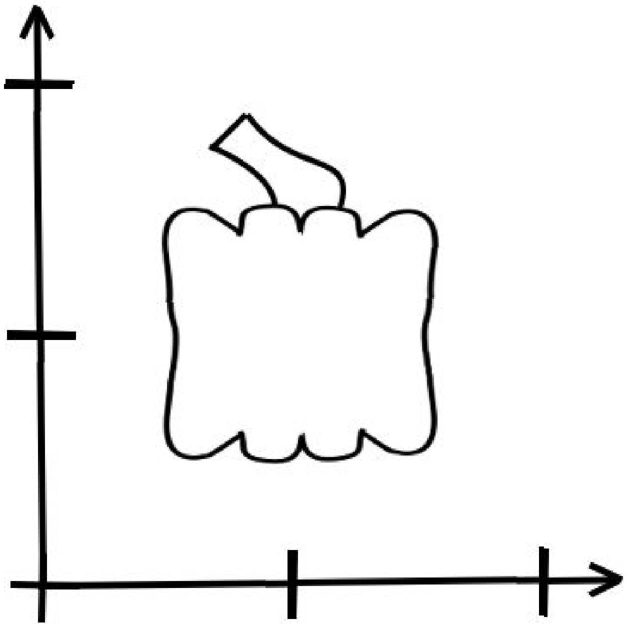
Given this pumpkin at (1,1), do the following:

model *= trans(x+2,y+2);

model *= rot$_z$(90);

model *= scale$_x$(-1);

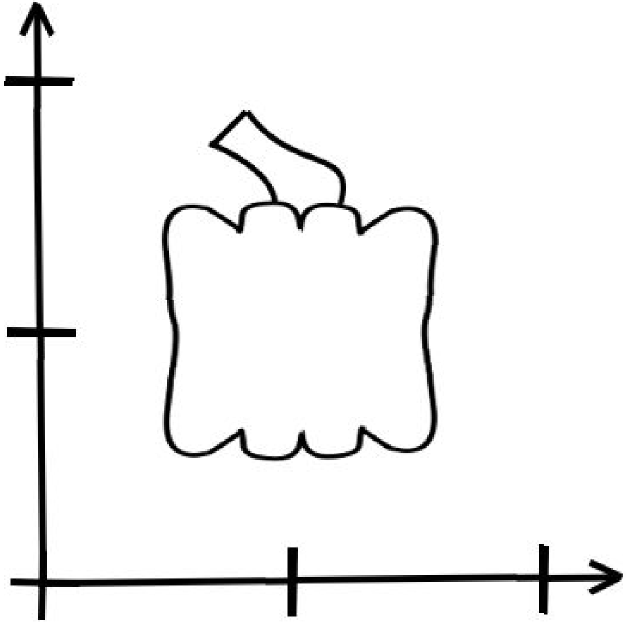
model *= trans(x-1,y-1);



Drawing example: Pumpkin

Given this pumpkin at (1,1), do the following:

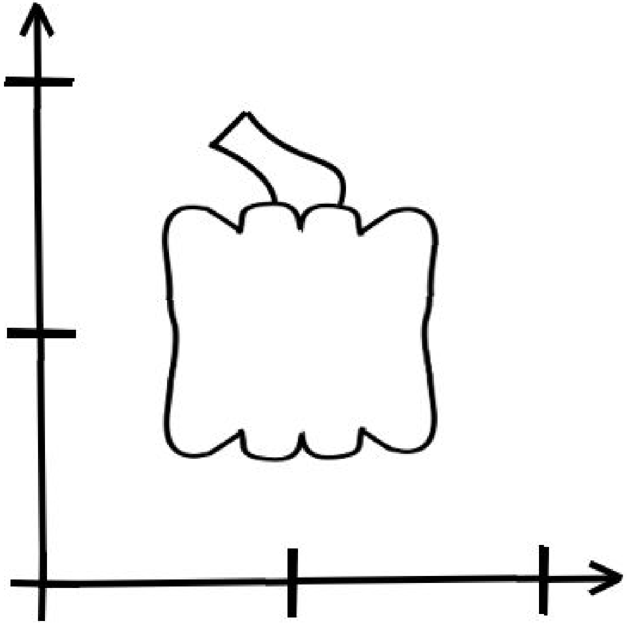
trans(2,2) * rot$_z$(90) * scale$_x$(-1) * trans(-1,-1)



Drawing example: Pumpkin

Manually writing the product of matrices

$\text{trans}(2,2) * \text{rot}_z(90) * \text{scale}_x(-1) * \text{trans}(-1,-1)$



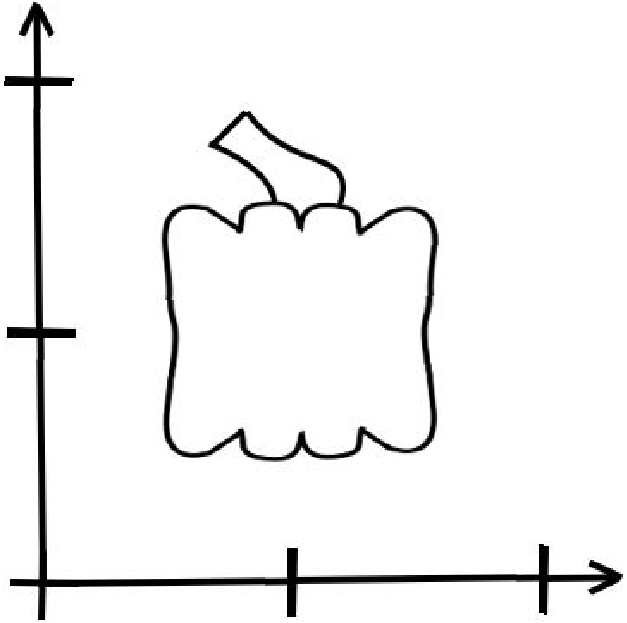
= what actual matrices?

Drawing example: Pumpkin

- Manually writing the product of matrices

trans(2,2) * rot$_z$(90) * scale$_x$(-1) * trans(-1,-1) = ?

- Multiply out the product with the "drawing below" trick
- Apply the final product to some points (0,0), (0,2), (2,0)

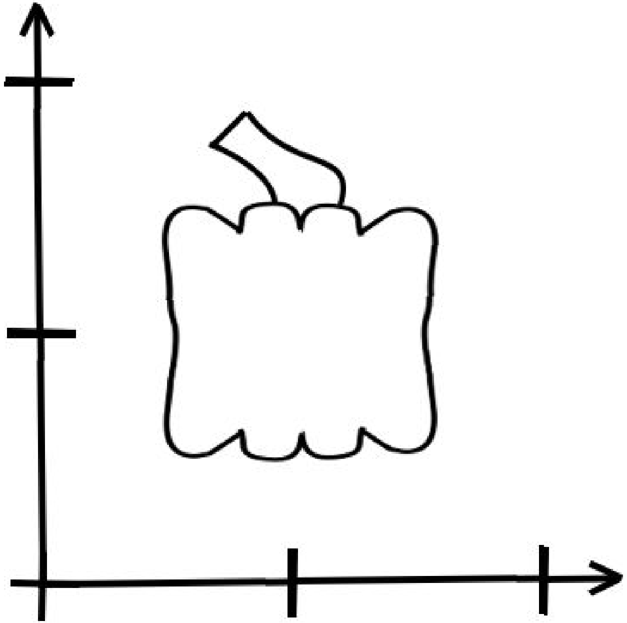


Drawing example: Pumpkin

- Actually draw out where the pumpkin moves at each step of

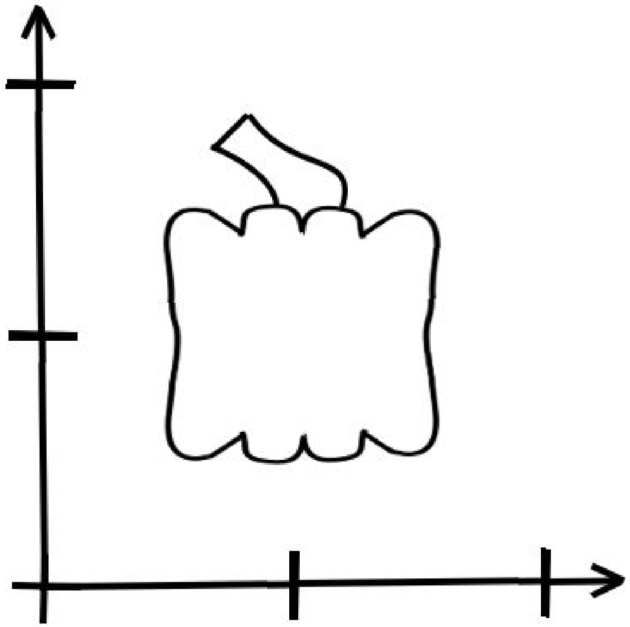
$\text{trans}(2,2) * \text{rot}_z(90) * \text{scale}_x(-1) * \text{trans}(-1,-1)$

- We’re treating it like an image -> Start at point and move Right-to-Left
- Show that where it landed is consistent with where the product displaced the 3 points to



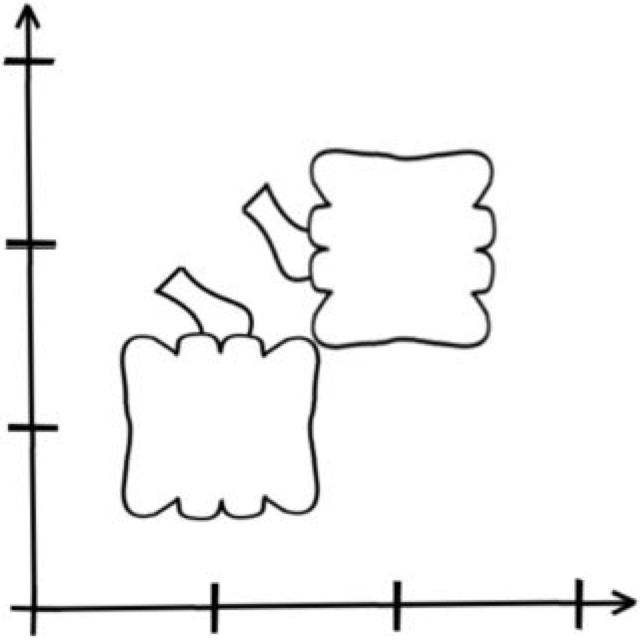
Drawing example: Pumpkin

- Actually draw out where a basis would move at each step (go left-right, maintain a basis as your temporary instead of a point)
- Wherever the origin winds up, draw the original image there using those axes

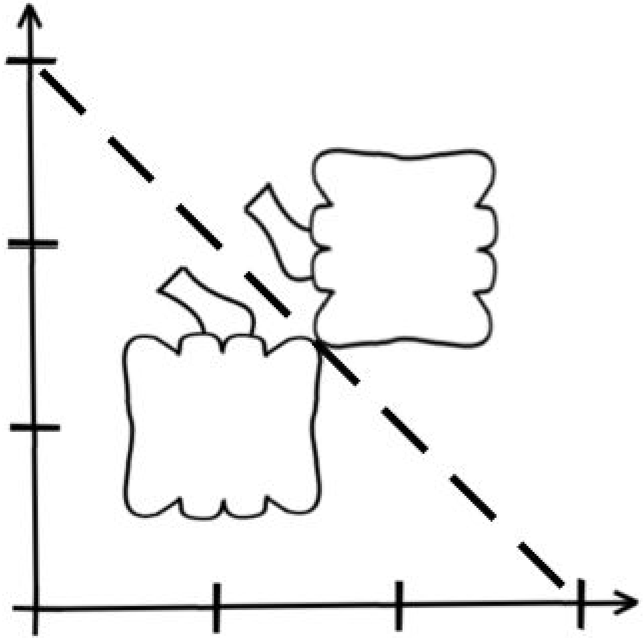


Drawing example: Pumpkin

- Why do we prefer left to right when building programs?
- Because of our temporary "partial matrices" when making the various products
  - Each sets us up for the next piece of a hierarchical model

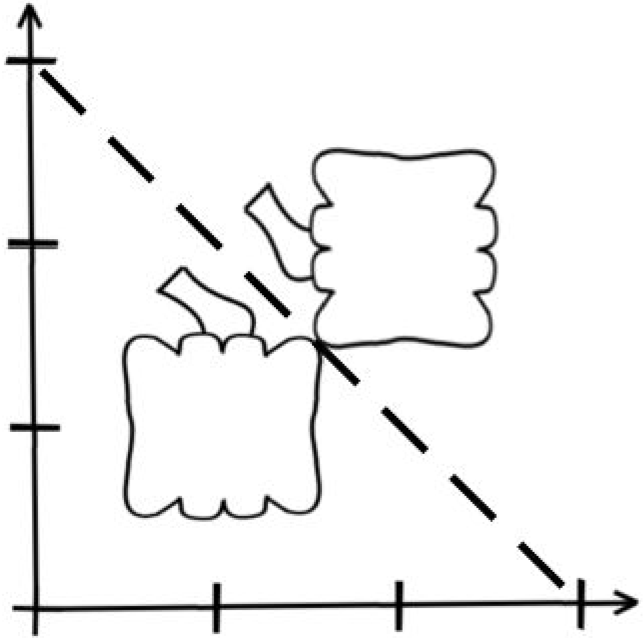


Checking our Answer



Checking our Answer

- Easily summarized as a reflection around a line from (3,0) to (0,3)
- The sequence of transforms to do that reflection is different:
  - trans(0,3) * rot$_z$(-45) * scale$_y$(-1) * rot$_z$(45)* trans(0,-3)
  - What's the code for this?
- Numerically multiplying it out, it was the same matrix, surprise!!!



Part V: Odd Transformation orders

Non-uniform scales

“Rotational shear” vs “rigid body” motion demo

- Accidental shearing problem
    - Non-uniform scales spell disaster next time a rotate gets postmultiplied
- “Rotate in oval” code demo

Replace your code in display() with:

```
 model_transform = mult( model_transform, scale( 5, 1, 1 ) );
 model_transform = mult( model_transform, rotation( this.graphicsState.animation_time/20, 0, 0, 1 ) );


 this.m_cube.draw( this.graphicsState, model_transform, new Material( vec4( .5,.5,.5,1 ), 1, 1, 1, 40, "earth.gif" ) );
CURRENT_BASIS_IS_WORTH_SHOWING(this, model_transform);
```

“Rotational shear” vs “rigid body” motion demo

- Rotate then Scale: Expected rigid-body movement
- Scale then Rotate: “shear rotation”
- What is rigid body?
  - Preserves angles
  - Which matrices do that?
- “Shear rotation” vs. plain shear matrix
  - plain shear approaches infinitely thin instead of rotating