universidade federal de pelotas
CDTec
centro de desenvolvimento tecnológico

computação
Universidade Federal de Pelotas

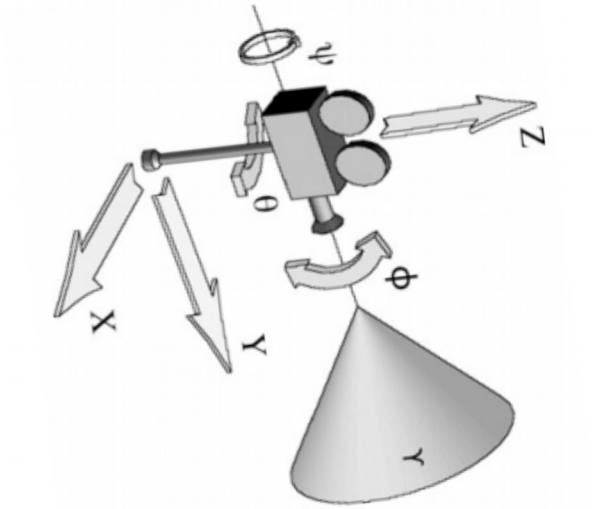
Computer Graphics
Today: Geometric Transformation

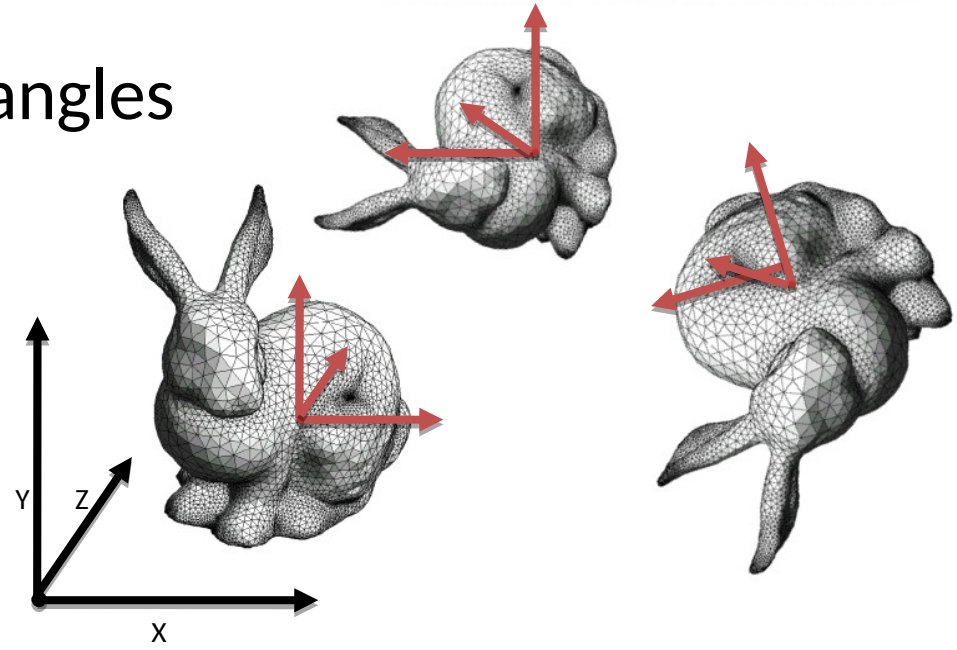
Prof. PhD Rafael P. Torchelsen
rafael.torchelsen@inf.ufpel.edu.br

Motivation

computação
Universidade Federal de Pelotas

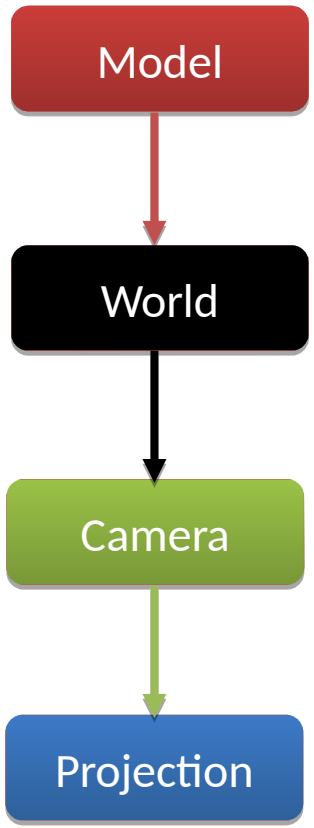
- Different coordinate systems
- Modeling
  - Place an object
  - Size
  - View it from different angles
- Animation

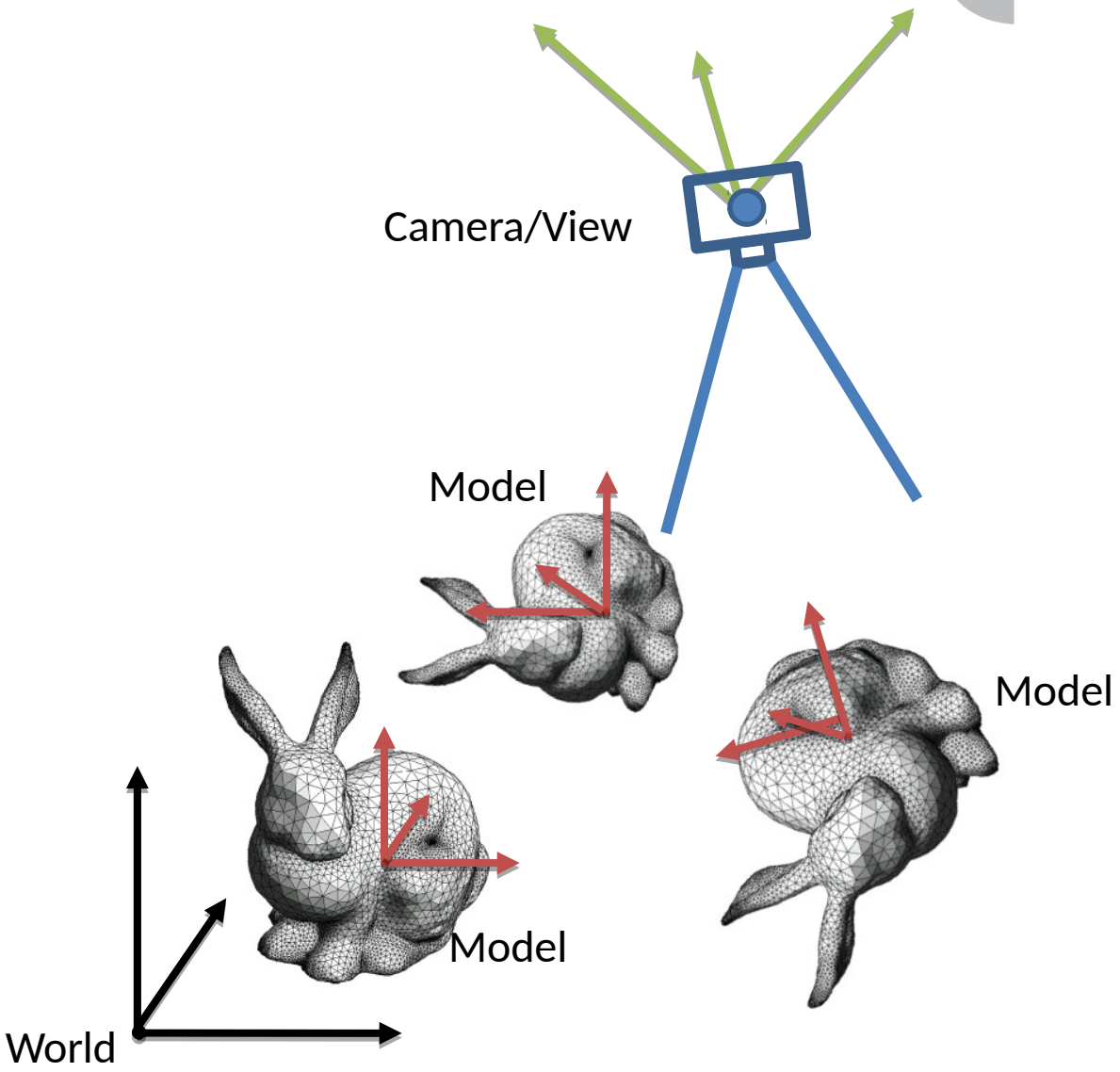




Each basis has a purpose

computação
Universidade Federal de Pelotas





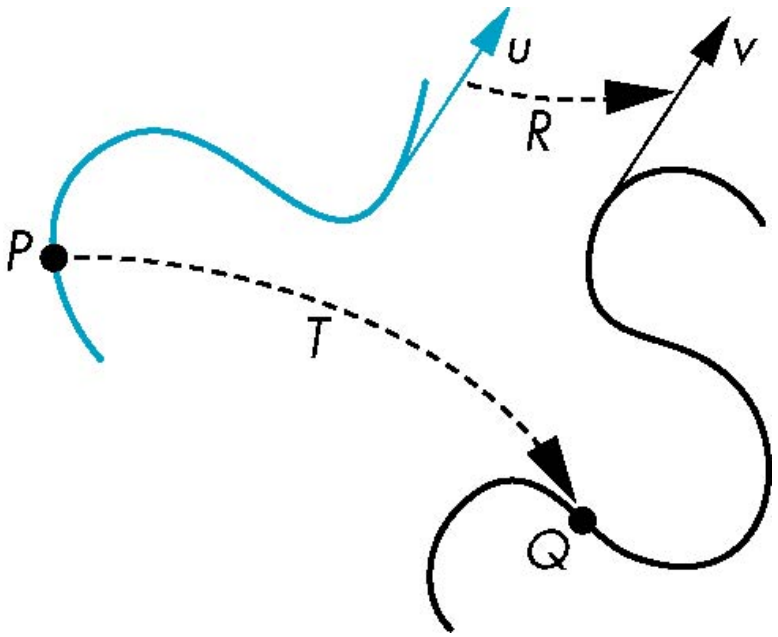
Transformations

computação
Universidade Federal de Pelotas

- Transformations 2D and 3D
  - Translation
  - Rotation
  - Scale
  - Shear
- Transformation classes
- Combination of transformations

Concept

computação
Universidade Federal de Pelotas

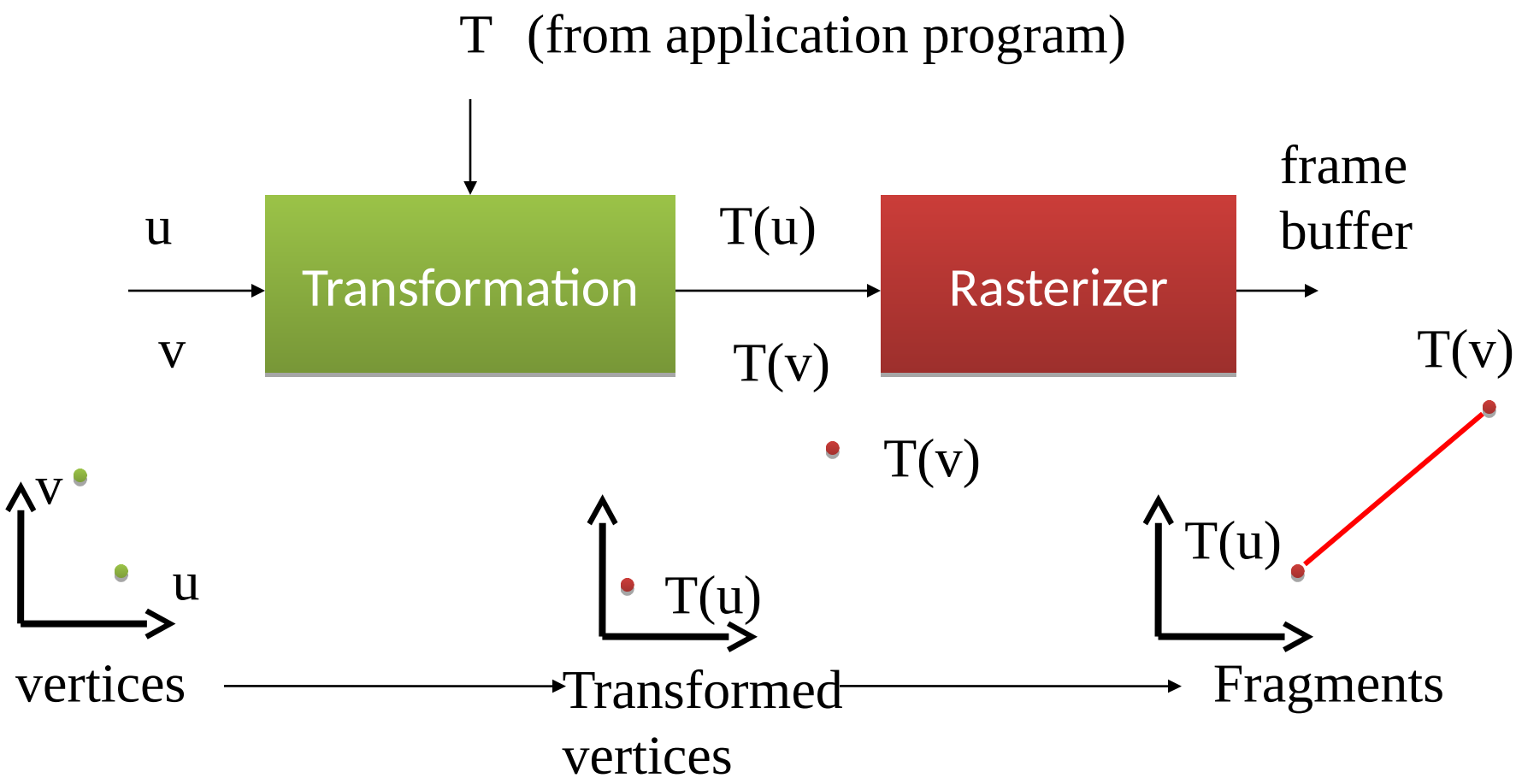


v=T(u)

A transformation **maps** points to other points and/or vectors to other vectors

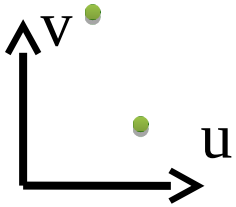
Pipeline Implementation

computação
Universidade Federal de Pelotas

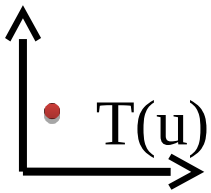


Current GPU **Graphics** Pipeline

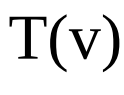
computação
Universidade Federal de Pelotas

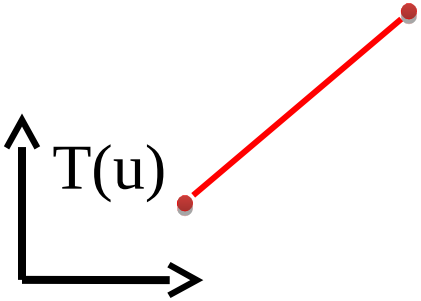


vertices

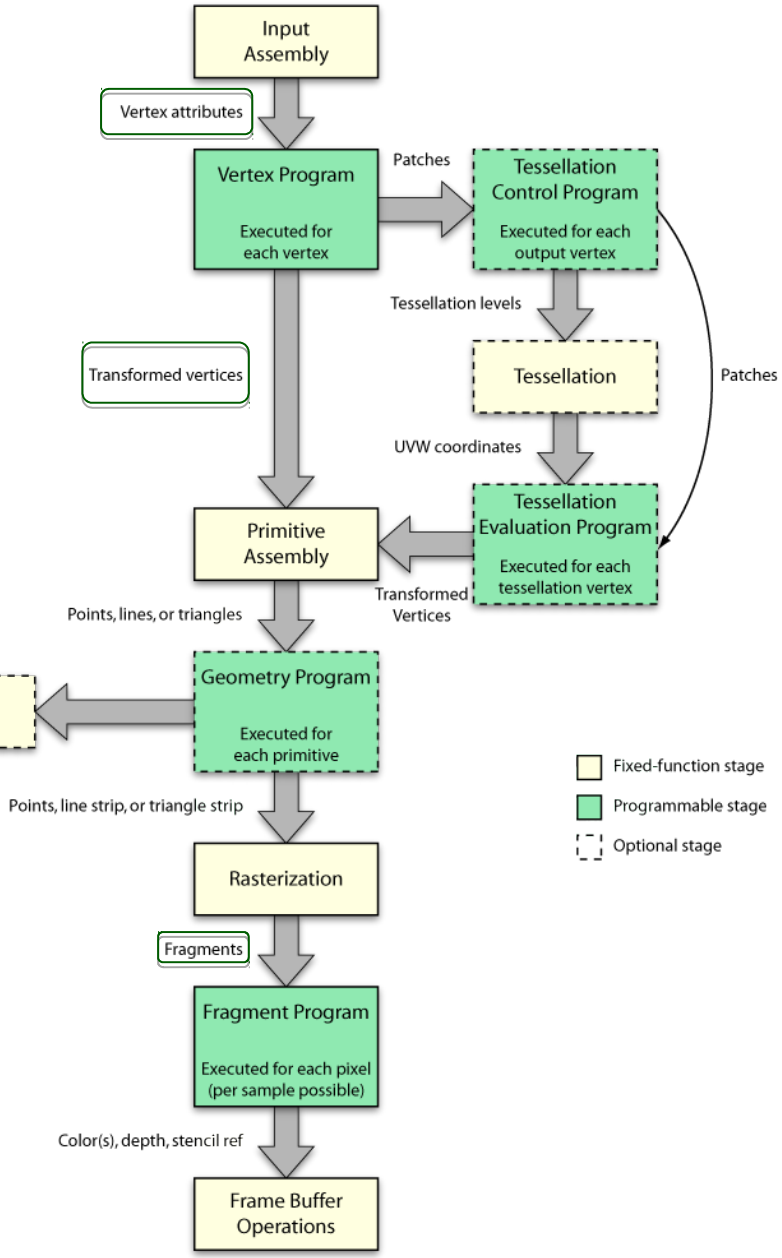


Transformed vertices





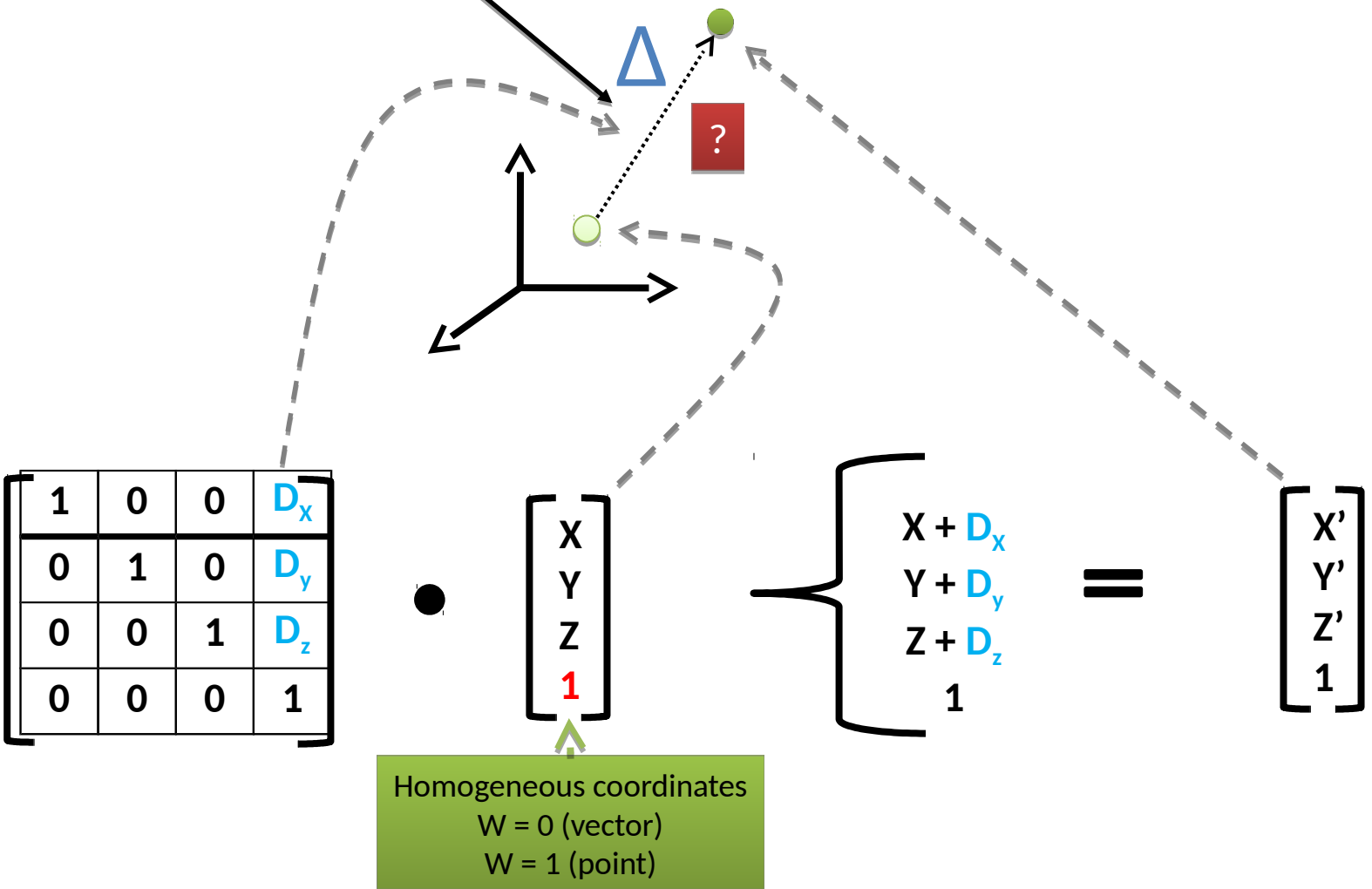
Fragments



Translation: Vertice

computação
Universidade Federal de Pelotas

How we compute this vector given two points?



Homogeneous coordinates

computação
Universidade Federal de Pelotas

Vector + Vector = Vector

$$\begin{bmatrix} U_x \\ U_y \\ U_z \\ 0 \end{bmatrix} + \begin{bmatrix} V_x \\ V_y \\ V_z \\ 0 \end{bmatrix} = \begin{bmatrix} U_x + V_x \\ U_y + V_y \\ U_z + V_z \\ 0 \end{bmatrix}$$

Point + Vector = Point

$$\begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} + \begin{bmatrix} V_x \\ V_y \\ V_z \\ 0 \end{bmatrix} = \begin{bmatrix} P_x + V_x \\ P_y + V_y \\ P_z + V_z \\ 1 \end{bmatrix}$$

Point - Point = Vector

$$\begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} - \begin{bmatrix} R_x \\ R_y \\ R_z \\ 1 \end{bmatrix} = \begin{bmatrix} P_x - R_x \\ P_y - R_y \\ P_z - R_z \\ 0 \end{bmatrix}$$

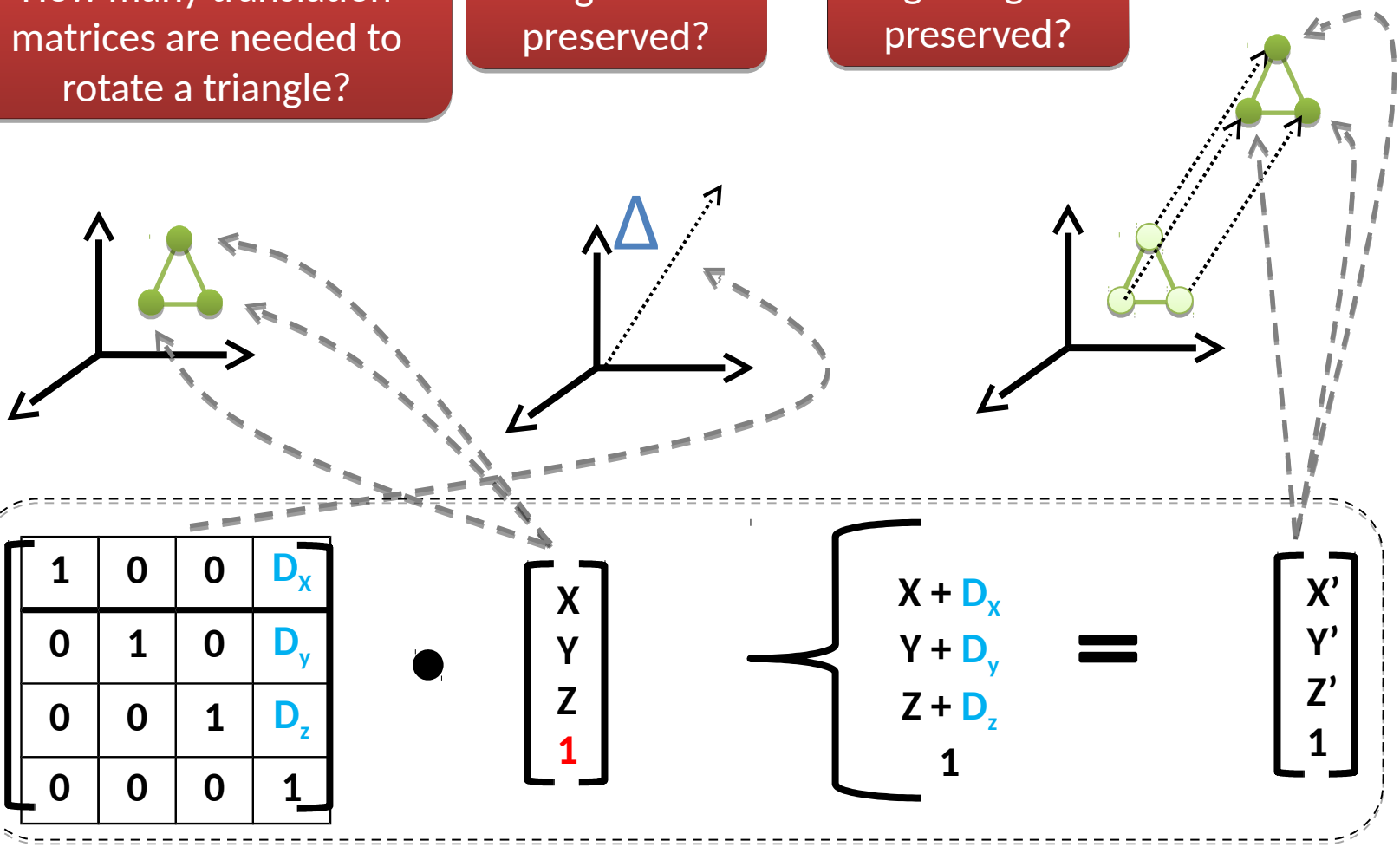
Translation: Mesh

computação
Universidade Federal de Pelotas

How many translation matrices are needed to rotate a triangle?

Angles are preserved?

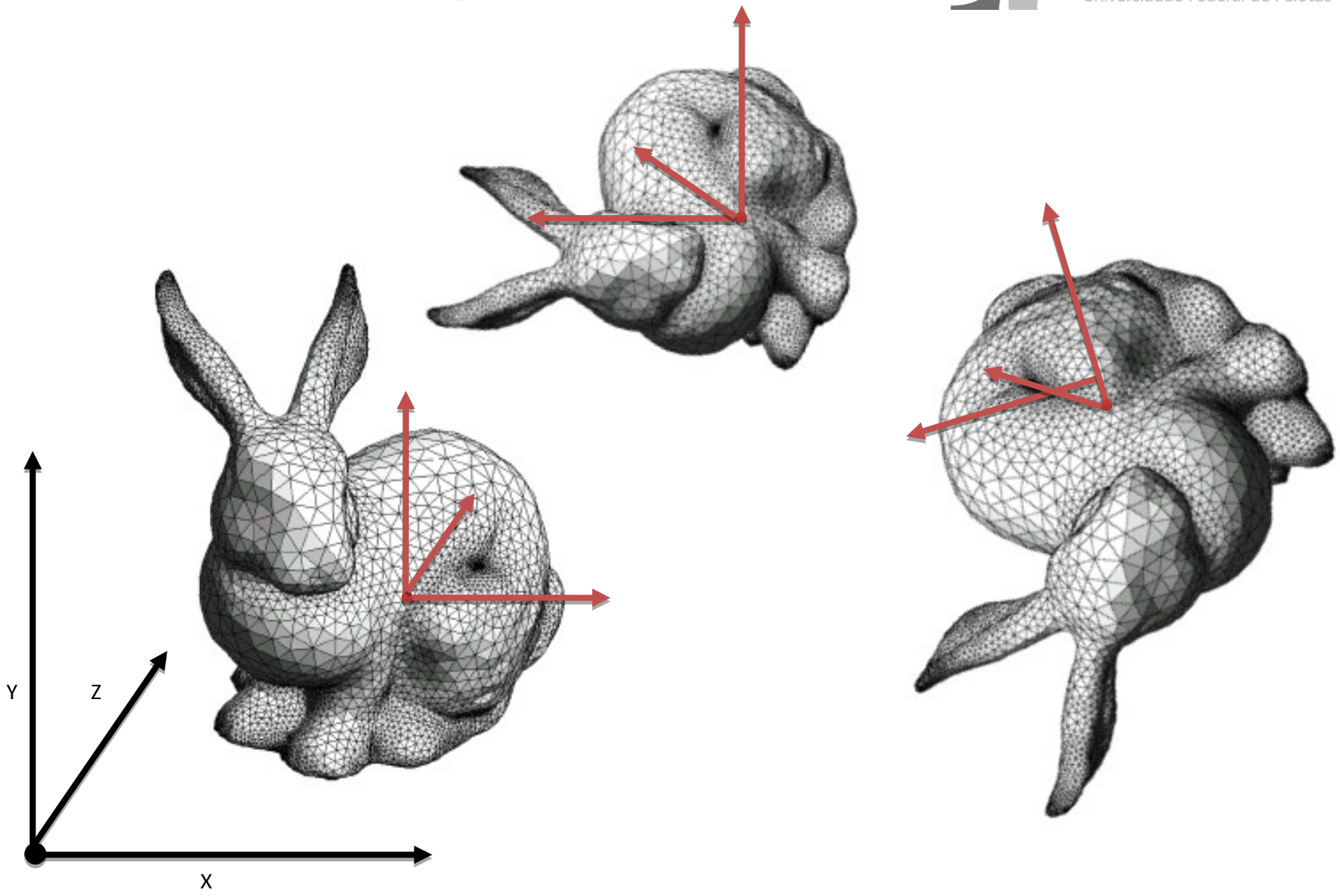
Edge length is preserved?



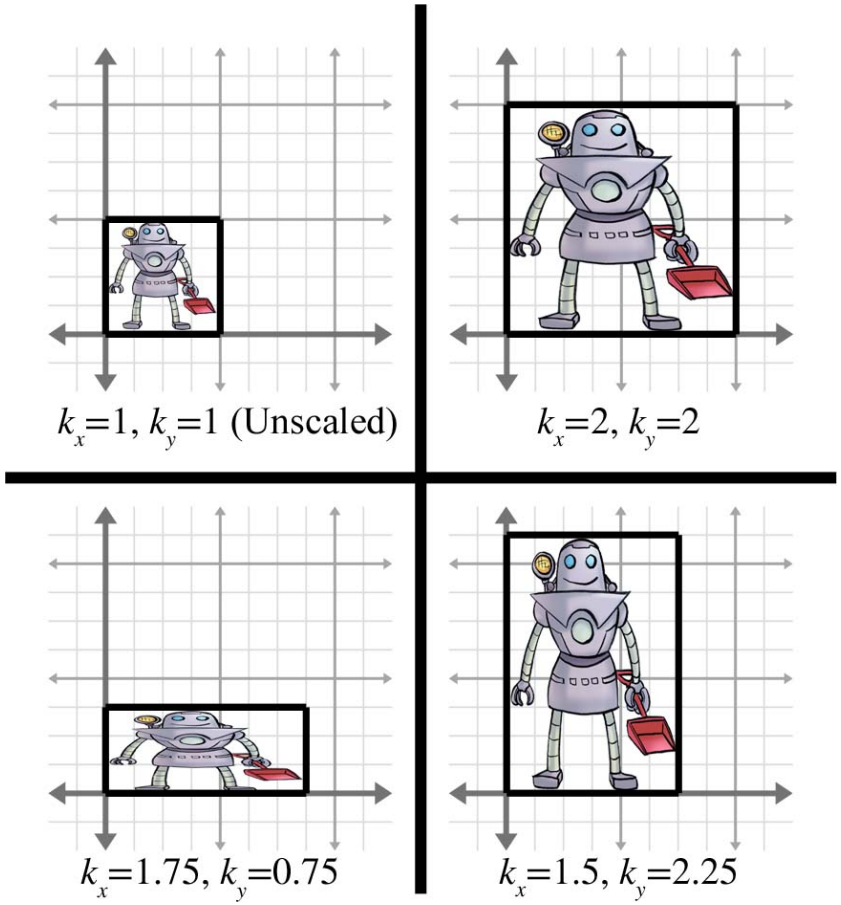
Per vertice

One matrix per model

computação
Universidade Federal de Pelotas



Scale



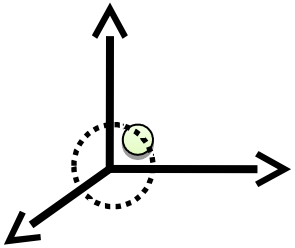
Scale

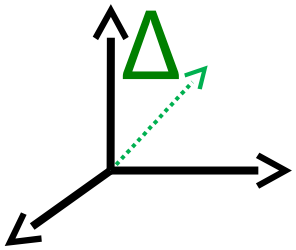
computação
Universidade Federal de Pelotas

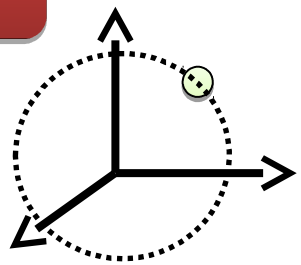
Edge length is preserved?

Angle is preserved?

Always?



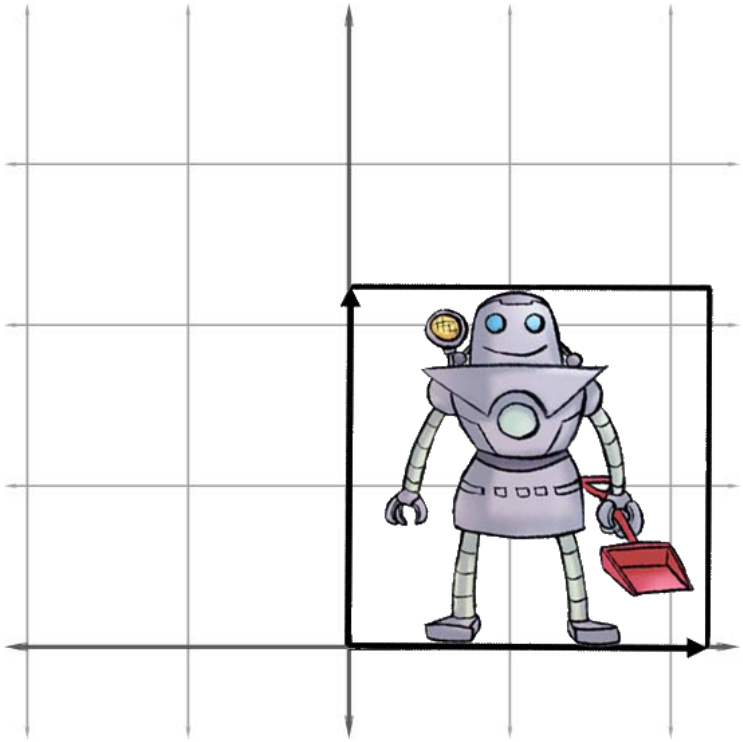




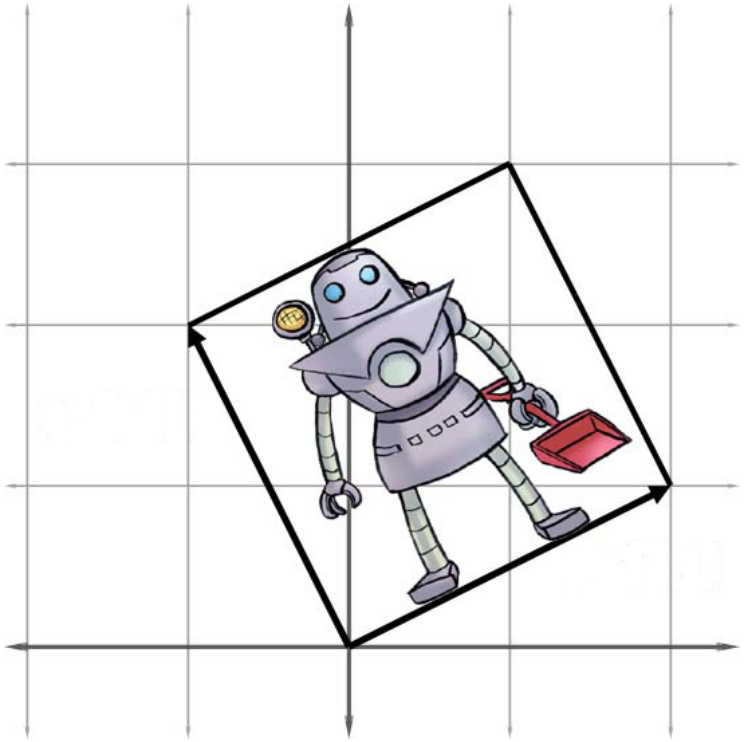
$$\begin{bmatrix} D_x & 0 & 0 & 0 \\ 0 & D_y & 0 & 0 \\ 0 & 0 & D_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \begin{cases} X \cdot D_x \\ Y \cdot D_y \\ Z \cdot D_z \\ 1 \end{cases} = \begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix}$$

Rotation

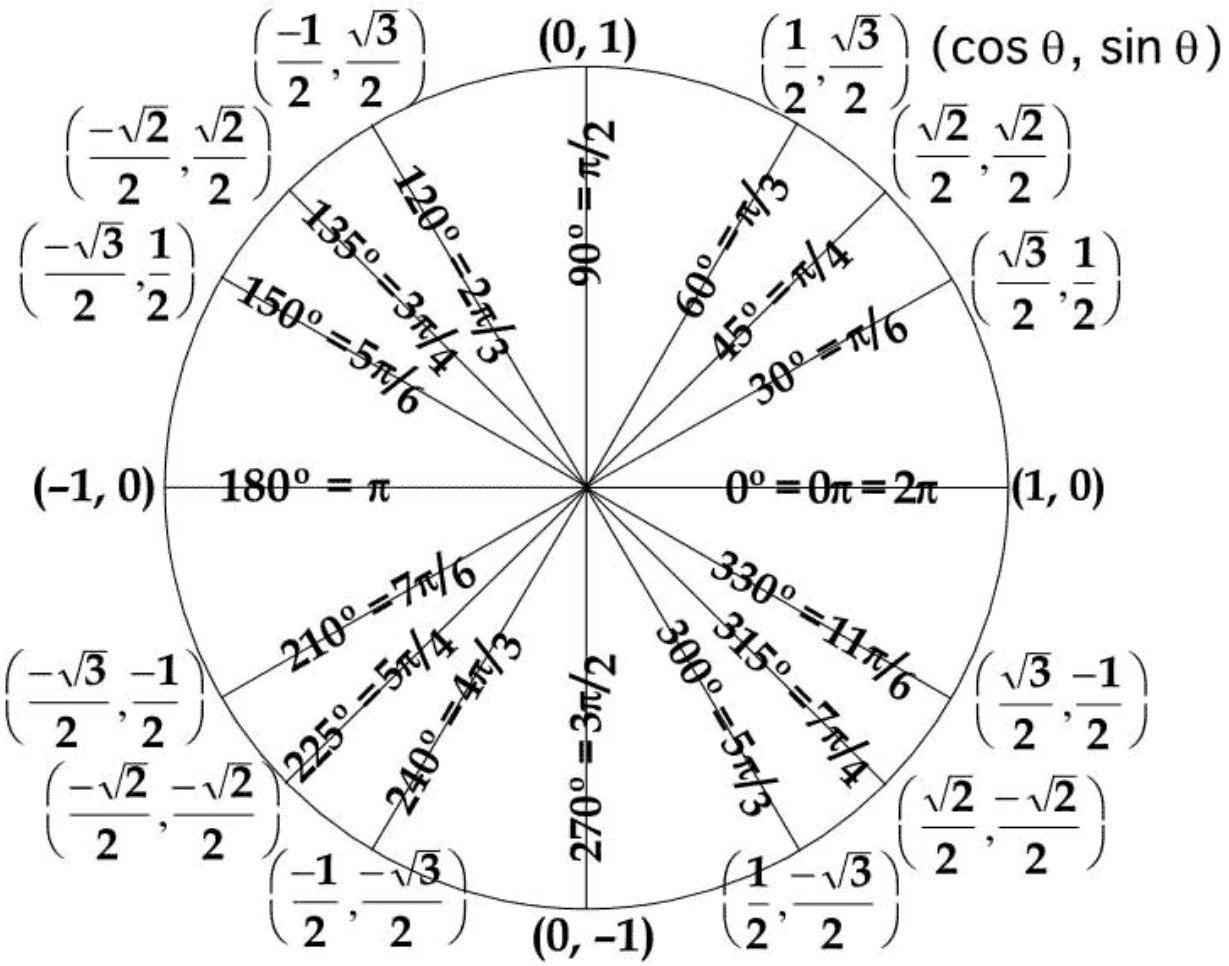
computação
Universidade Federal de Pelotas



Before



After



computação
de Federal de Pelotas

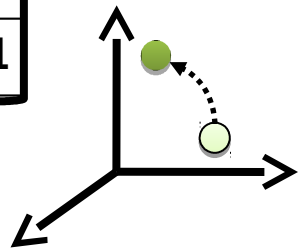
Rotation around which axis?

X

0	Sα	Cα	0
0	0	0	1

Y

Cα	0	Sα	0
0	1	0	0
-Sα	0	Cα	0
0	0	0	1



Z

Cα	-Sα	0	0
Sα	Cα	0	0
0	0	1	0
0	0	0	1

$\bullet$

X
Y
Z
1

X · Cα + Y · -Sα
X · Sα + Y · Cα
Z
1

$=$

X'
Y'
Z'
1

Inverses

computação
Universidade Federal de Pelotas

- **Translation**: $\mathbf{T}^{-1}(d_x, d_y, d_z) = \mathbf{T}(-d_x, -d_y, -d_z)$
- **Rotation**: $\mathbf{R}^{-1}(\theta) = \mathbf{R}(-\theta)$
  - Holds for any rotation matrix
  - Note that since $\cos(-\theta) = \cos(\theta)$ and $\sin(-\theta) = -\sin(\theta)$

  $\mathbf{R}^{-1}(\theta) = \mathbf{R}^{T}(\theta)$
- **Scaling**: $\mathbf{S}^{-1}(s_x, s_y, s_z) = \mathbf{S}(1/s_x, 1/s_y, 1/s_z)$

Are these operation invertible?

Yes, except scale = 0

Transformations

computação
Universidade Federal de Pelotas

- **Affine**: preserves ratio of areas
  - Translation
  - Rotation
  - Non-uniform or uniform scaling
  - Shearing

- **Conformal**: preserves angles
  - Translation
  - Rotation
  - Uniform scaling

- **Isometric**: preserves length
  - Translation
  - Rotation

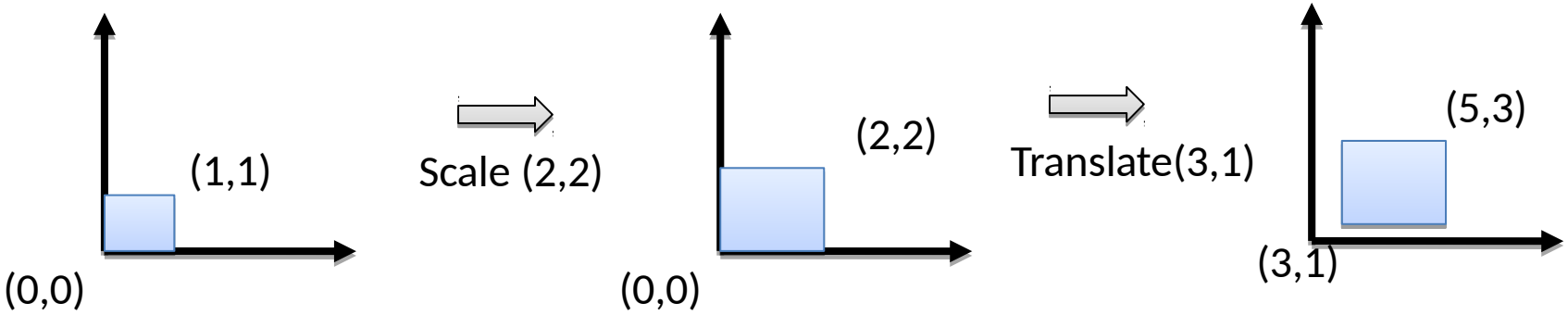
Concatenation

computação
Universidade Federal de Pelotas

- We can form arbitrary **affine transformation** matrices by multiplying together rotation, translation, and scaling matrices
- Because the same transformation is applied to many vertices, the cost of forming a matrix $\mathbf{M}=\mathbf{ABCD}$ is **not significant compared** to the cost of computing $\mathbf{Mp}$ for many vertices $\mathbf{p}$

Concatenation

computação
Universidade Federal de Pelotas



1	0	3
0	1	1
0	0	1

Translate(3,1)

2	0	0
0	2	0
0	0	1

Scale (2,2)

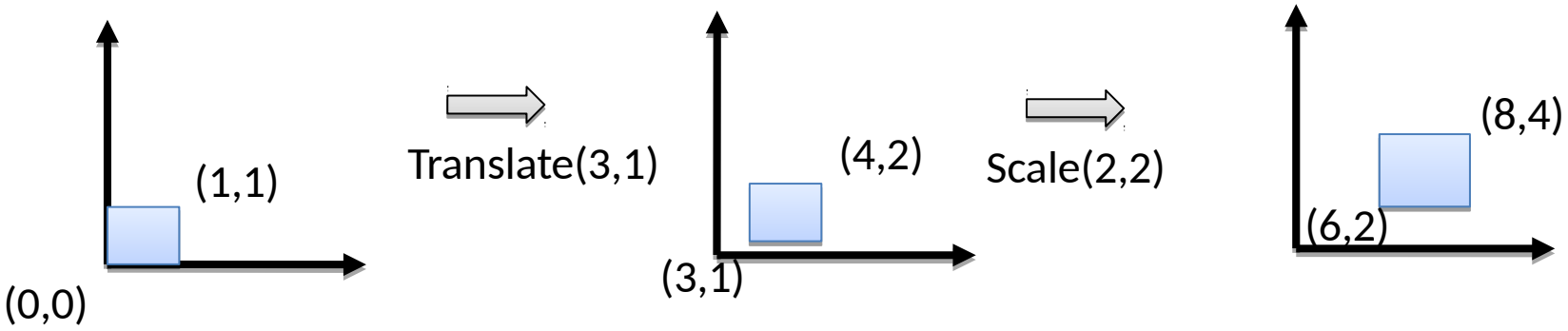
=

2	0	3
0	2	1
0	0	1

TS

Concatenation

computação
Universidade Federal de Pelotas



2	0	0
0	2	0
0	0	1

Scale (2,2)

1	0	3
0	1	1
0	0	1

Translate(3,1)

=

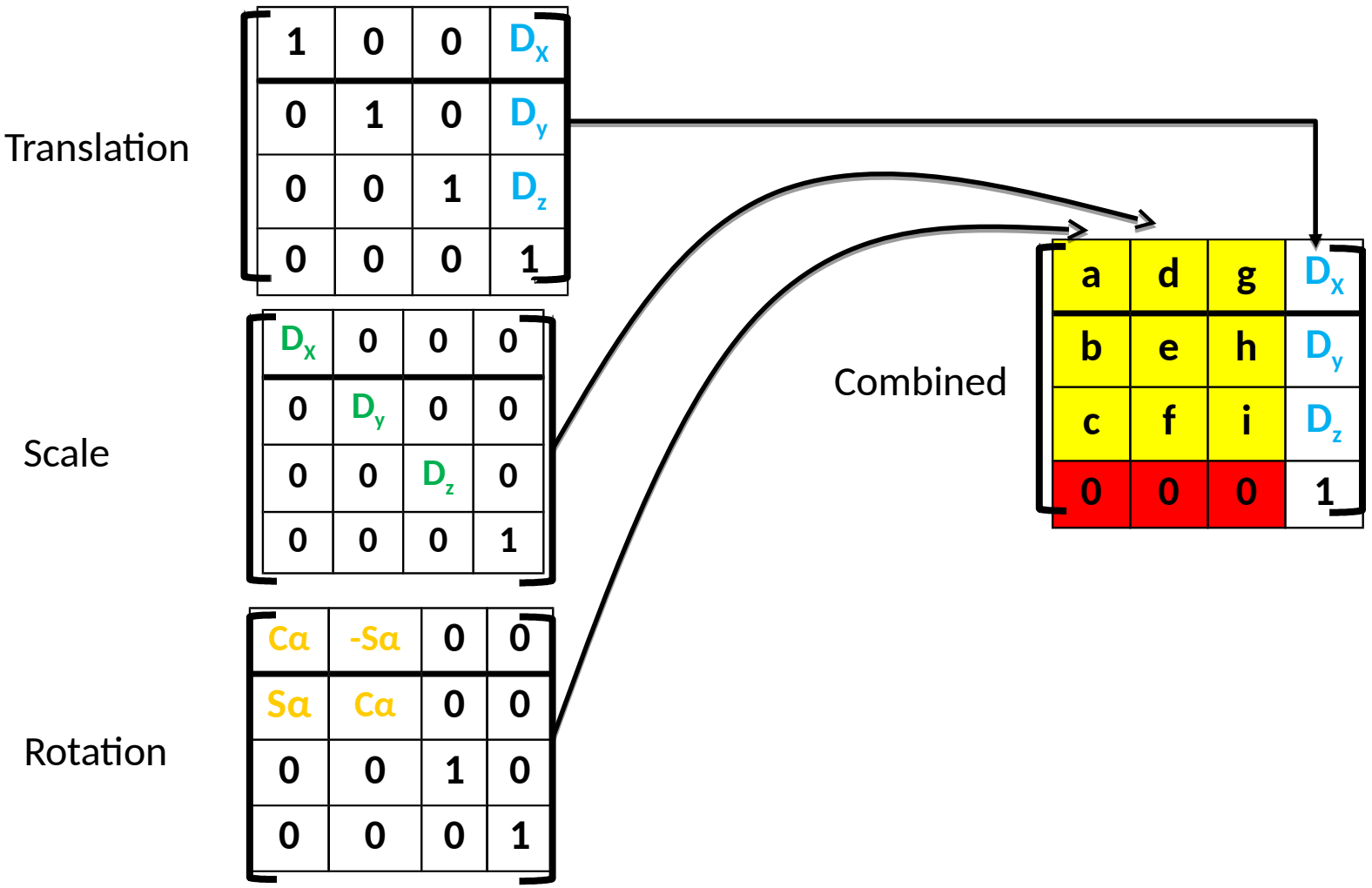
2	0	6
0	2	2
0	0	1

ST

Which one is the correct?
It's application dependent

Concatenation

computação
Universidade Federal de Pelotas



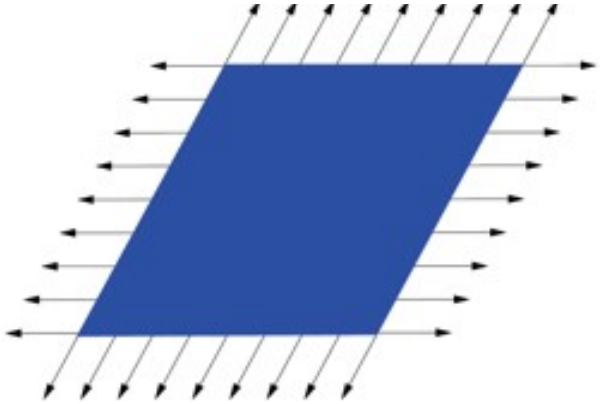
Normal transformation

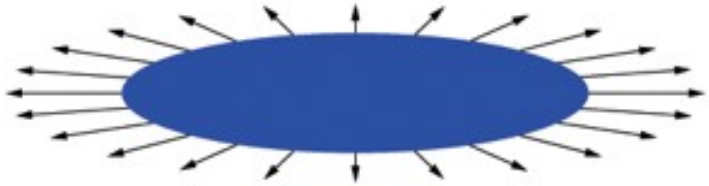
computação
Universidade Federal de Pelotas

Shear

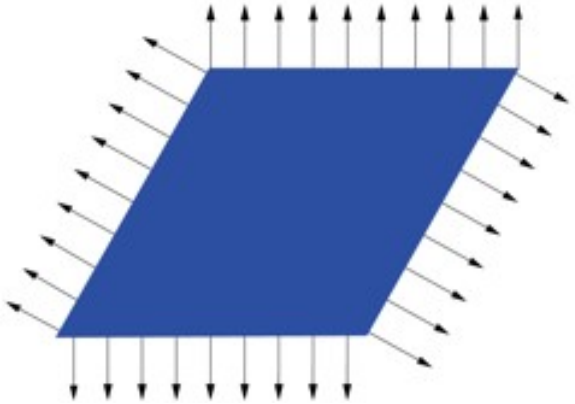
Scale

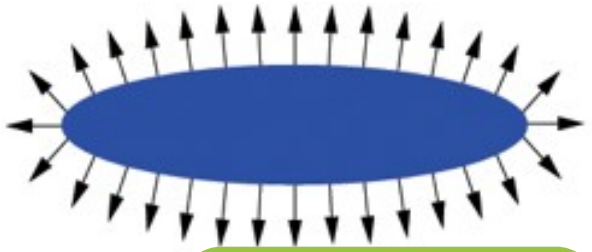
Incorrect Normal Transformation





Correct Normal Transformation





Bonus: 0,5
Explain how normals are transformed

Only one student, first to appear in the forum. Complete lesson!

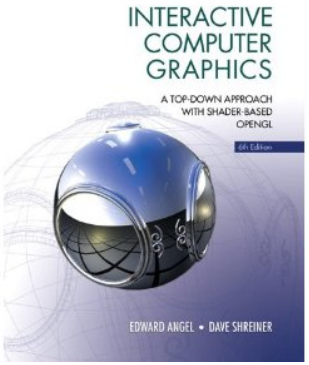
OpenGL

computação
Universidade Federal de Pelotas

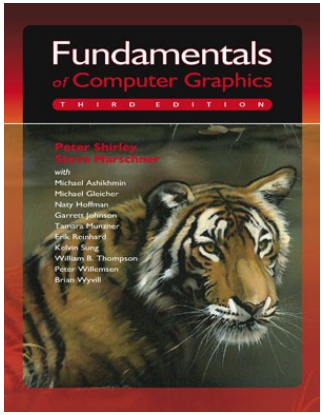
https://learnopengl.com/Getting-started/Transformations

Books

computação
Universidade Federal de Pelotas



Chapter 3



Chapter 6

Chapter 3 and 4

Read at least one!

Links

http://www.cs.princeton.edu/~gewang/projects/darth/stuff/quat_faq.html
http://www.realtimerendering.com/#xforms
http://www.geometrictools.com/
http://mathworld.wolfram.com/
http://www.gamedev.net
http://solarianprogrammer.com/2013/05/22/opengl-101-matrices-projection-view-model/