ARLAB

ME/CprE/ComS 557

Computer Graphics and Geometric Modeling

Navigation in 3D

October 29th, 2015

Rafael Radkowski

IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

This Thursday

- Homework presentations
- Every team has 10 minutes to present the homework
- All other in the room evaluate the presentation
- Additional credits, they do not count as homework points
- Make sure that all of you are here.
- Off-campus students: prepare a video.

Content

- Loading objects from files

Navigation in 3D scenes

- Introduction
- Ray-Intersection Test
- Code Example
- Window System and Keyboard Interaction
- Interpolation
- Scissor Test

Loading obj Files

3D Model File Formats

Files on hard disk that keep the 3D model information (i.e., vertices, relations, surface normal vectors, etc.)

- 3ds 3D Studio
- dae COLLADA
- fbx Autodesk FBX
- iv Open Inventor format
- obj Alias Wavefront
- stl Stereolithography file
- wrl Virtual Reality Modeling Language (VRML)

3D Model File Formats

VRAC | HCI

All file formats have been developed for a specific purpose. Thus, they are differences regarding their capabilities and provided features:

Differences between file formats:

- Binary vs. ASCII
  - Binary: fast to load, small file size
  - ASCII: human readable, file handling is easier for programmers, portable to different applications
- Content: what type of data does the file store
  - meshes (OpenGL primitives)
  - NURBS
  - 2D images
  - Color and Material
  - Animations
- Amount of data
  - Total number of primitives
  - File size limitation

Learn the specification of the 3D model files you want to use!

Alias Wavefront (.obj)

obj file format was developed as for Wavefront's 3D software. It stores geometric objects as meshes in ASCII format (using the ".obj" file extension) or in binary format (using the .MOD extension). The binary format is proprietary and undocumented. Due to its simplicity, obj has become a famous file format.

Today, its usage is limited due to the limited capabilities.

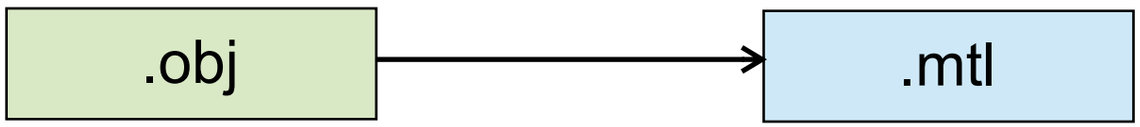
It supports

- lines, polygons, as well as free-form curves and surfaces.
- Lines and polygons are described in terms of their points
- Curves and surfaces are defined with control points and other information depending on the type of curve.
- Color and Material
- Textures (image information is not part of the file format)

Know your file format. Many problems occur during format conversion.

http://www.fileformat.info/format/wavefrontobj/egff.htm

Alias Wavefront (.obj)



.obj keeps the geometric information (structure of object)

.mtl keeps the color, material, and texture information (appearance of object)

The file format uses symbols (v, vt, f, etc.) to identify the geometric items.

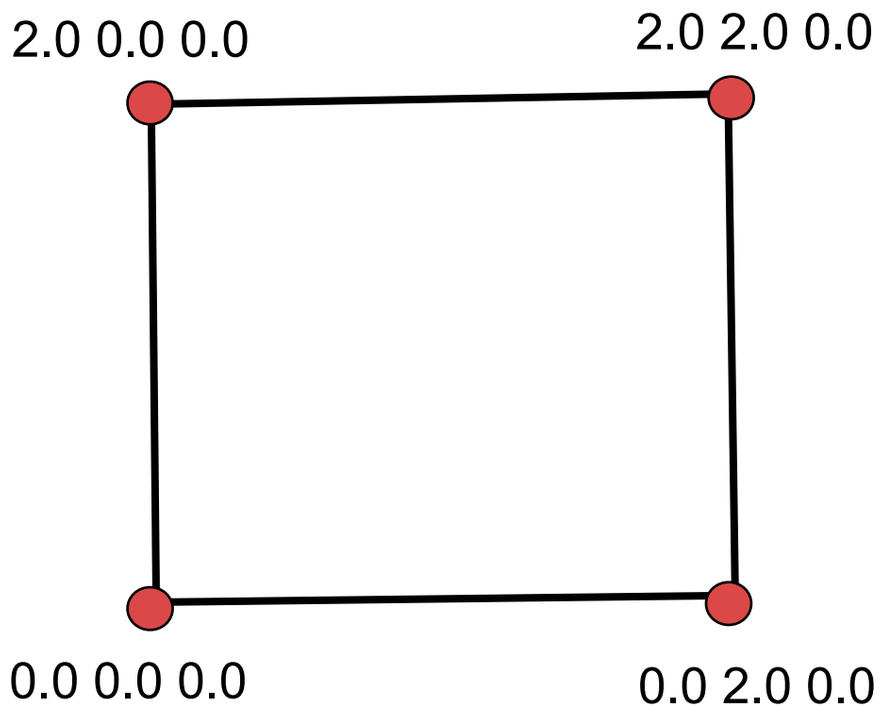
Example

```
# A 2 x 2 square mapped
mtllib master.mtl
v  0.000000 2.000000 0.000000
v  0.000000 0.000000 0.000000
v  2.000000 0.000000 0.000000
v  2.000000 2.000000 0.000000
vt 0.000000 1.000000
vt 0.000000 0.000000
vt 1.000000 0.000000
vt 1.000000 1.000000
# 4 vertices
usemtl wood
# The first number is the point, then the slash,
# and the second is the texture point
f 1/1 2/2 3/3 4/4
# 1 element
```

Material file

Vertices

Face



Alias Wavefront (.obj)

VRAC | HCI

Vertex data:

v	Geometric vertices:	**v** *x y z*
vt	Texture vertices:	**vt** *u v*
vn	Vertex normals:	**vn** *dx dy dz*

Elements:

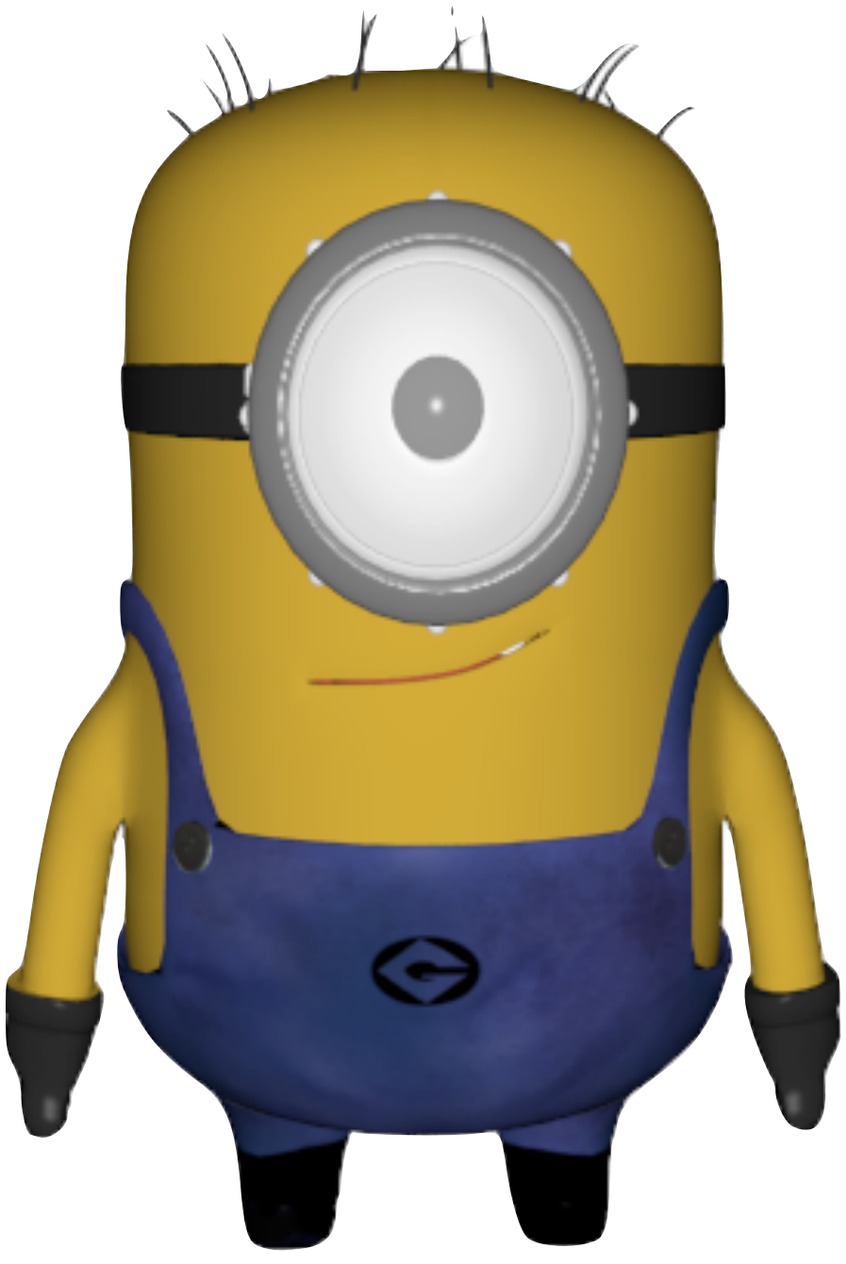
p	Point:	**p** *v1*
l	Line:	**l** *v1 v2 ... vn*
f	Face:	**f** *v1 v2 ... vn*
f	Face with texture coords:	**f** *v1/t1 v2/t2 .... vn/tn*
f	Face with vertex normals:	**f** *v1//n1 v2//n2 .... vn//nn*
f	Face with txt and norms:	**f** *v1/t1/n1 v2/t2/n2 .... vn/tn/nn*

Display/render attributes:

usemtl	Material name:	**usemtl** *materialname*
mtllib	Material library:	**mtllib** *materiallibname.mtl*

bold: the command symbol, italic: the data

Example



Minion.obj

Minion.obj

```
1  # WaveFront *.obj file (generated by Cheetah3D)
2
3  mtllib Minion.mtl
4
5  g Cylinder01
6  v 0.026487 0.024717 -0.006314
7  v 0.026565 0.026507 -0.008578
8  v 0.026385 0.024717 -0.008563
9  v 0.026667 0.026507 -0.006314
10 v 0.026260 0.026507 -0.010825
11 v 0.026083 0.024717 -0.010794
12 v 0.026341 0.028360 -0.010839
13 v 0.026646 0.028360 -0.008585
14 v 0.026749 0.028360 -0.006314
15 v 0.025758 0.026507 -0.013038
16 v 0.025583 0.024717 -0.012992
17 v 0.025059 0.026507 -0.015199
18 v 0.024890 0.024717 -0.015139
19 v 0.025137 0.028360 -0.015227
20 v 0.025837 0.028360 -0.013058
21 v 0.025098 0.032101 -0.015213
22 v 0.025843 0.030237 -0.013060
23 v 0.025797 0.032101 -0.013048
24 v 0.025142 0.030237 -0.015229
25 v 0.026348 0.030237 -0.010840
26 v 0.026301 0.032101 -0.010832
27 v 0.026653 0.030237 -0.008586
28 v 0.026605 0.032101 -0.008582
29 v 0.026755 0.030237 -0.006314
30 v 0.026708 0.032101 -0.006314
31 v 0.024169 0.026507 -0.017292
32 v 0.024005 0.024717 -0.017218
33 v 0.023094 0.026507 -0.019302
34 v 0.022938 0.024717 -0.019215
35 v 0.023165 0.028360 -0.019342
36 v 0.024243 0.028360 -0.017326
37 v 0.021845 0.026507 -0.021214
38 v 0.021697 0.024717 -0.021113
39 v 0.020430 0.026507 -0.023012
40 v 0.020292 0.024717 -0.022900
41 v 0.020493 0.028360 -0.023064
42 v 0.021912 0.028360 -0.021260
43 v 0.020462 0.032101 -0.023038
44 v 0.021918 0.030237 -0.021264
45 v 0.021879 0.032101 -0.021237
46 v 0.020498 0.030237 -0.023068
47 v 0.023171 0.030237 -0.019346
48 v 0.023130 0.032101 -0.019322
49 v 0.024249 0.030237 -0.017329
50 v 0.024206 0.032101 -0.017309
51 v 0.018859 0.026507 -0.024683
```

76738 lines in file

Read obj files

The files

- GLObjectObj.h
- GLObjectObj.cpp

implement a simple obj object loader for triangle objects.

Usage:

Include the header file:

```
#include "GLObjectObj.h"
```

Code Example:

```
GLObjectObj* myObject = new GLObjectObj("path_and_file");
```

GLObjectObj

ARLAB

An API to load OBJ models from files.

The loader is limited to objects with triangles.

Constructor:

```
GLObjectObj(string path_and_file);
```

Methods:

Set an appearance object.

```
void setApperance(GLAppearance appearance)
```

- appearance - a GLAppearance object.

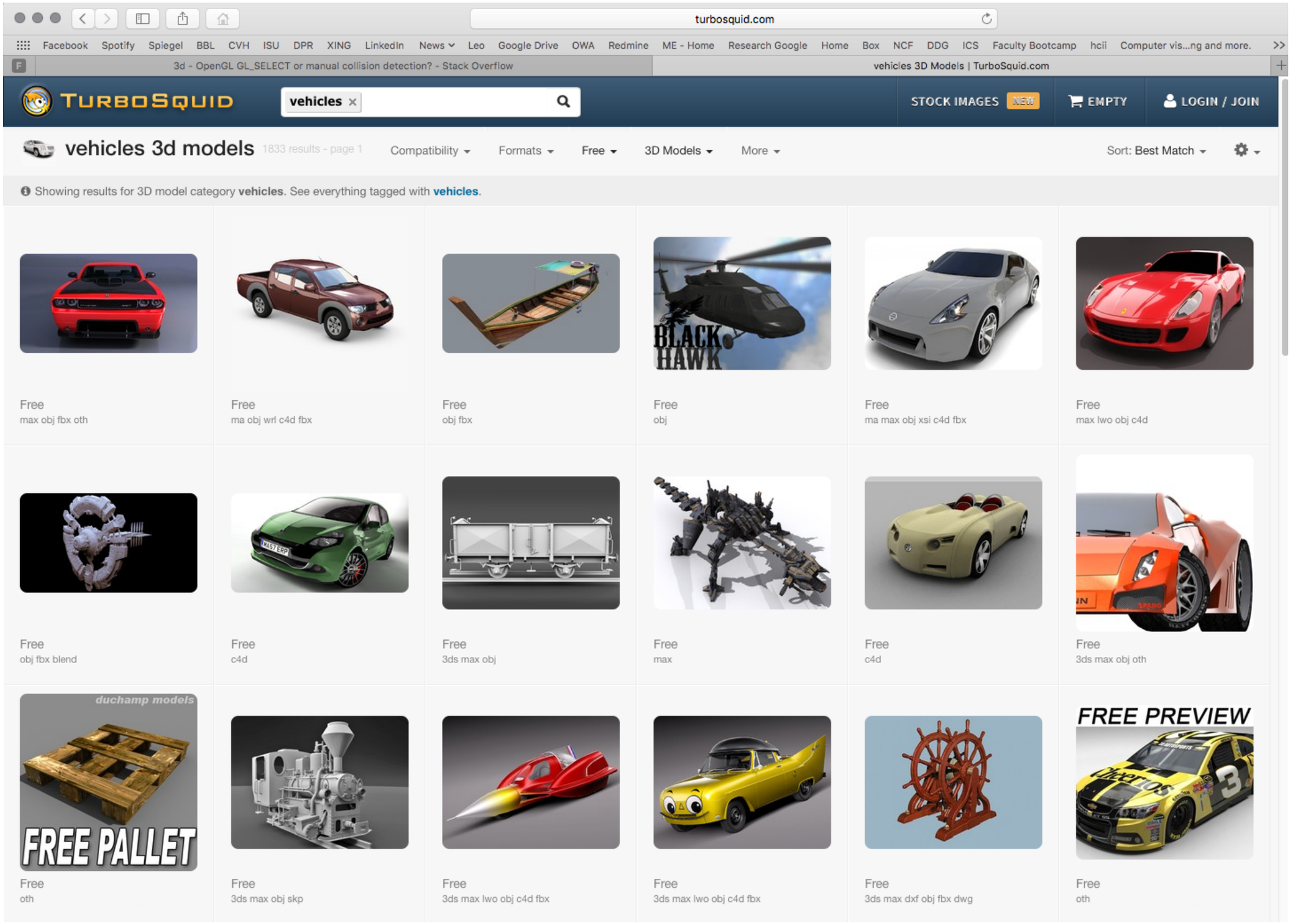
Initialize the object. MUST be called before one can use it.

```
void init(void)
```

Draw the object

```
void draw(void)
```

Turbosquid

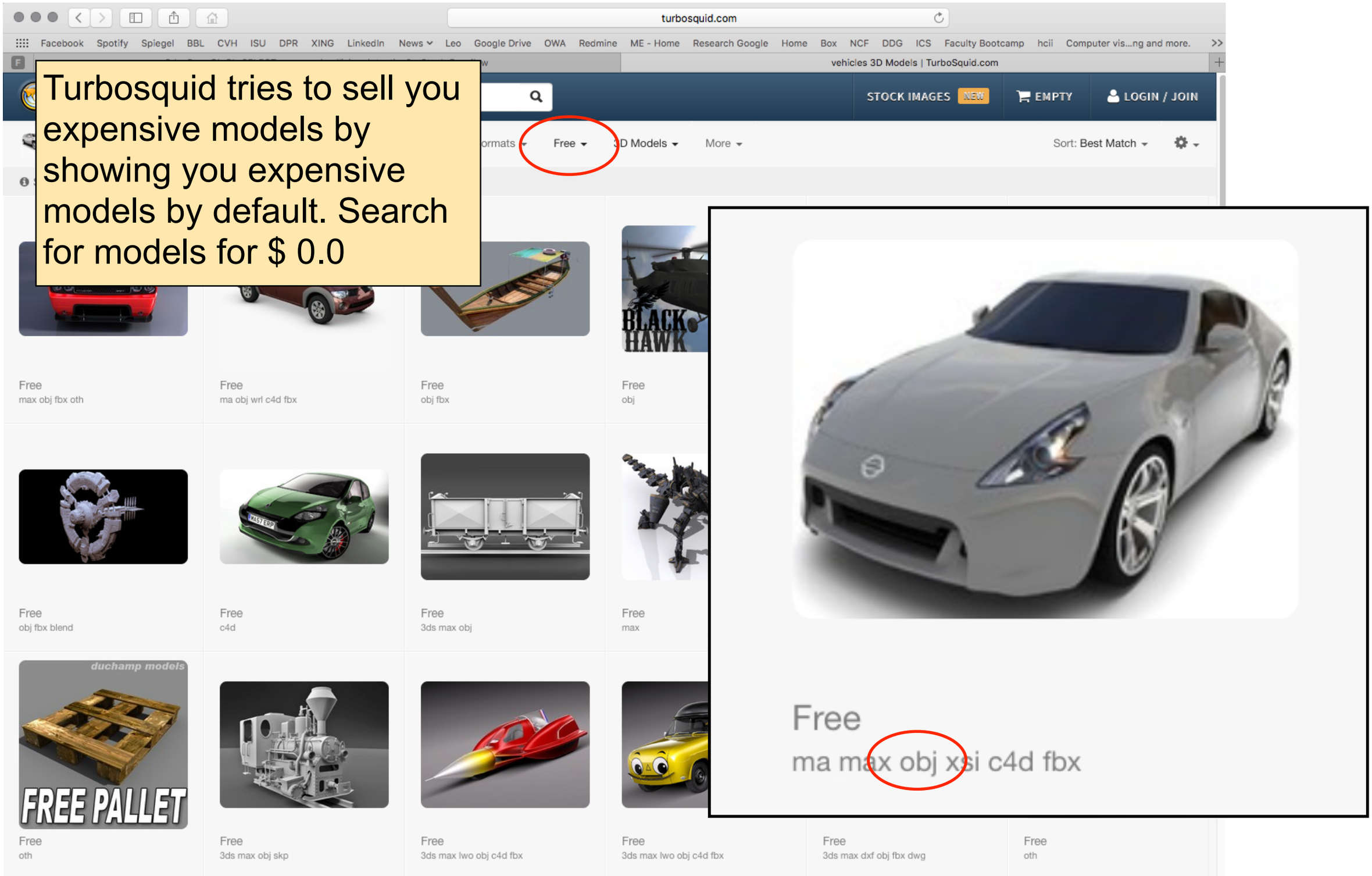


http://www.turbosquid.com

VRAC|HCI

IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

Turbosquid



Turbosquid tries to sell you expensive models by showing you expensive models by default. Search for models for $ 0.0

http://www.turbosquid.com

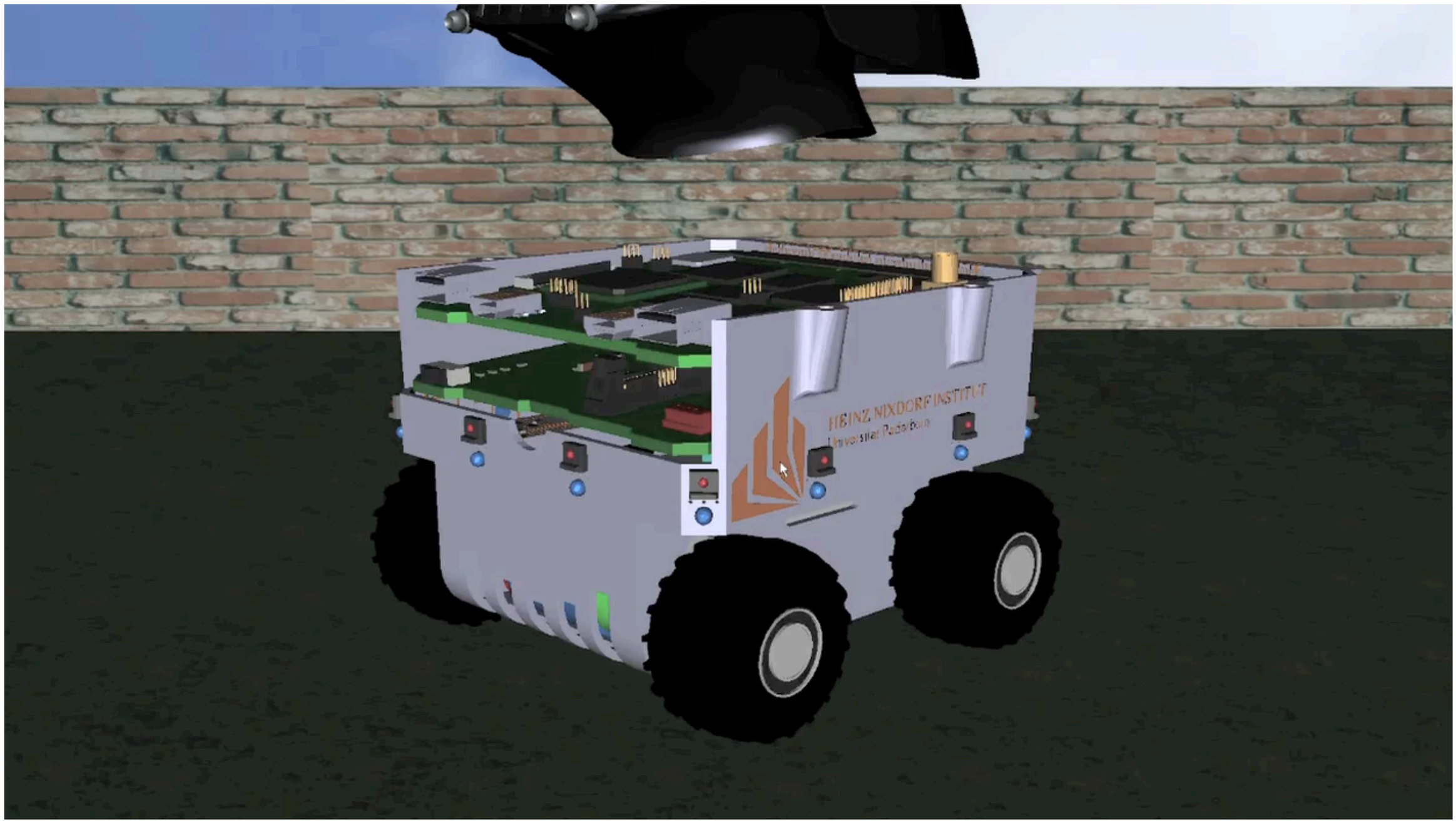
ARLAB

Introduction

VRAC|HCI

IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

ARLAB

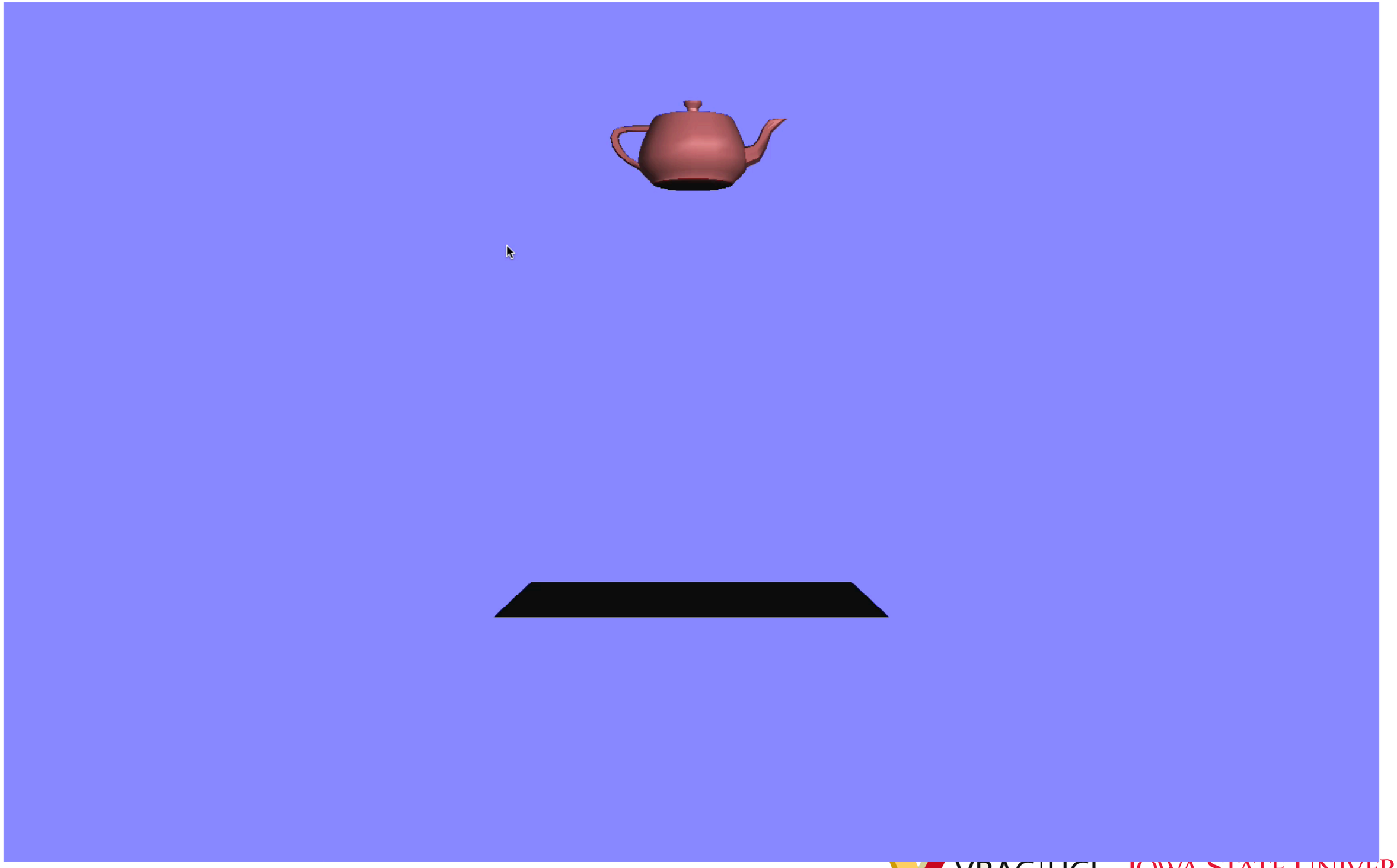


VRAC|HCI

IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

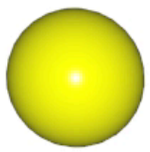
Video

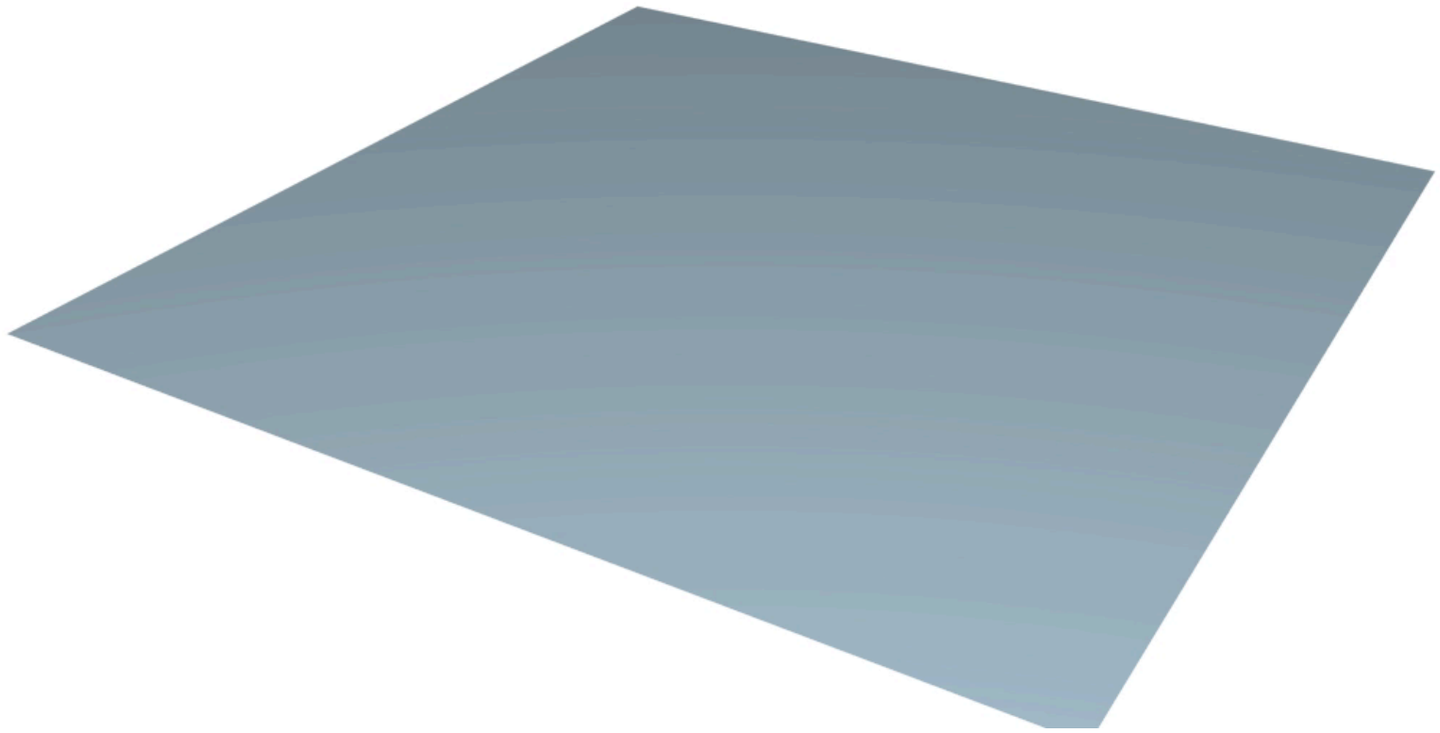
ARLAB



Video

ARLAB

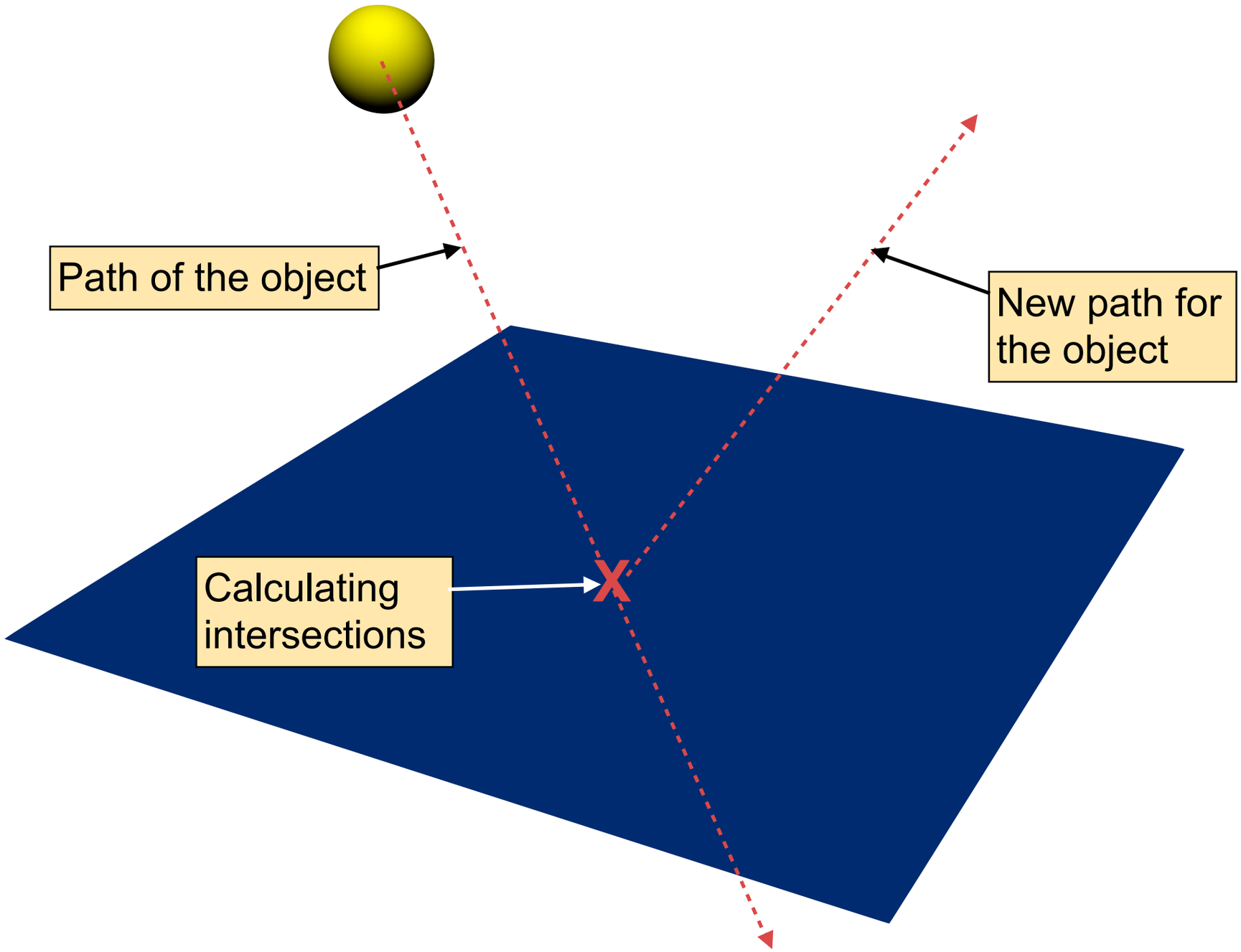




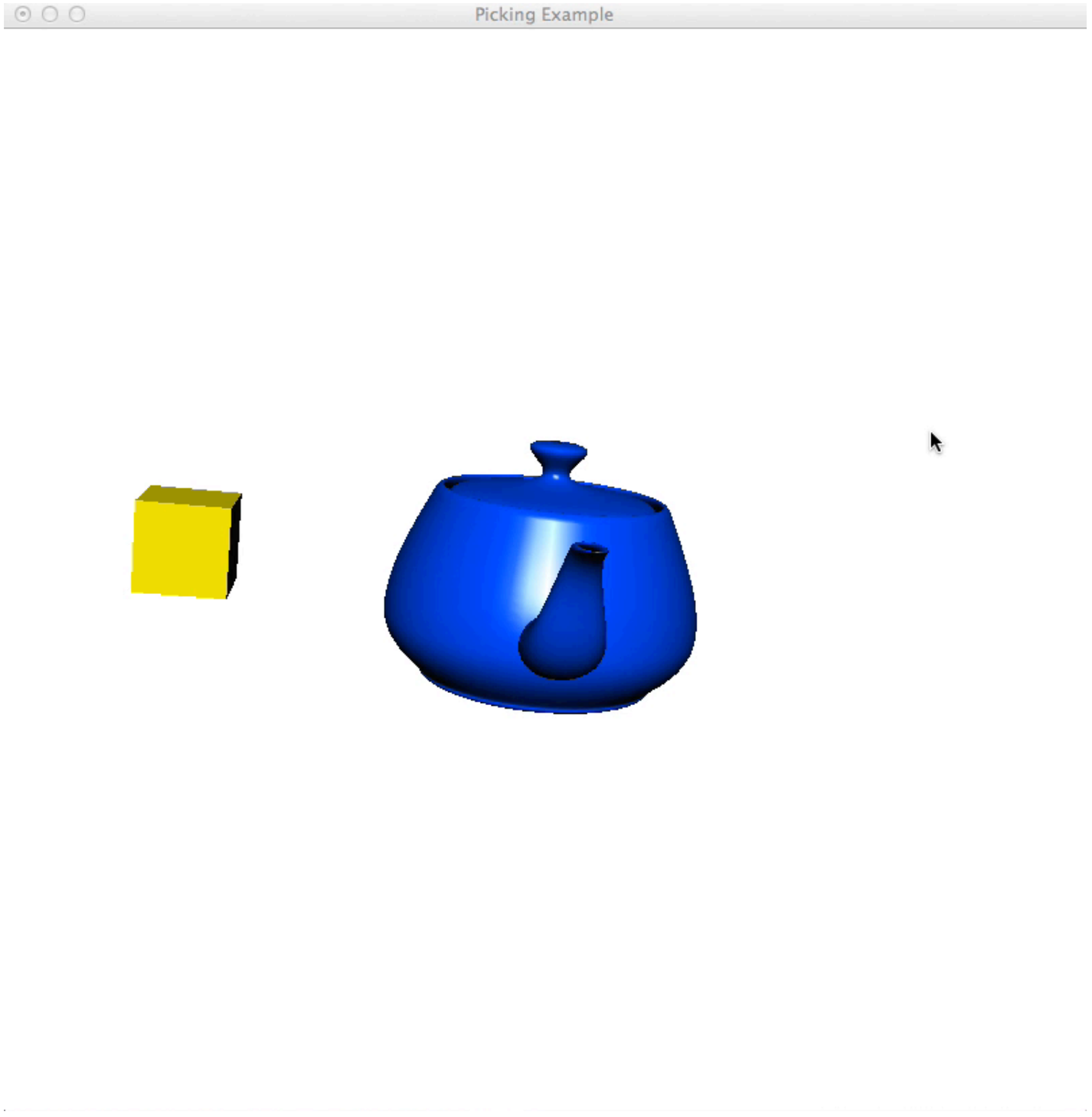
VRAC|HCI

IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

Introduction

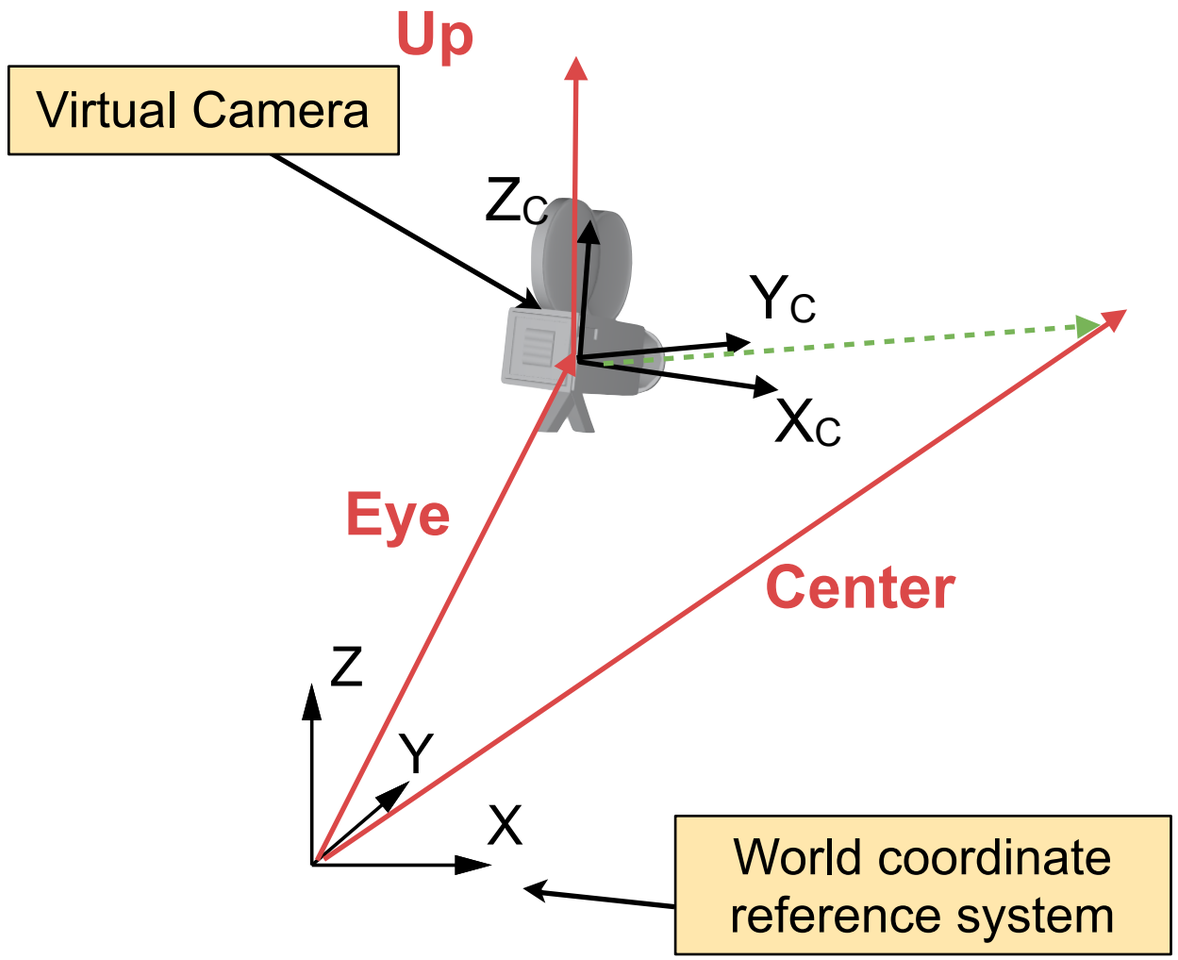


Video



View Matrix

- Camera Location and Orientation



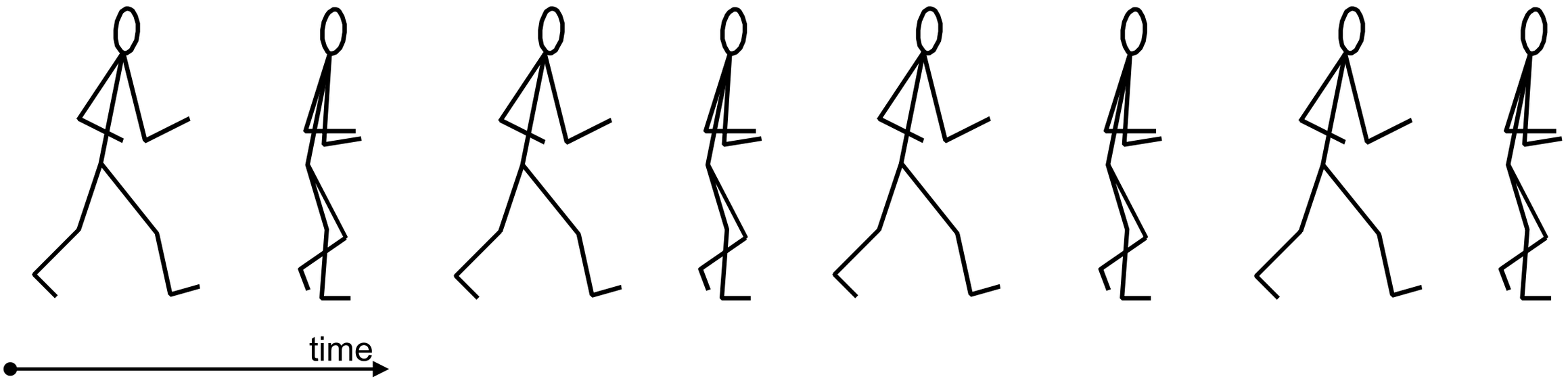
Vectors to setup a virtual camera

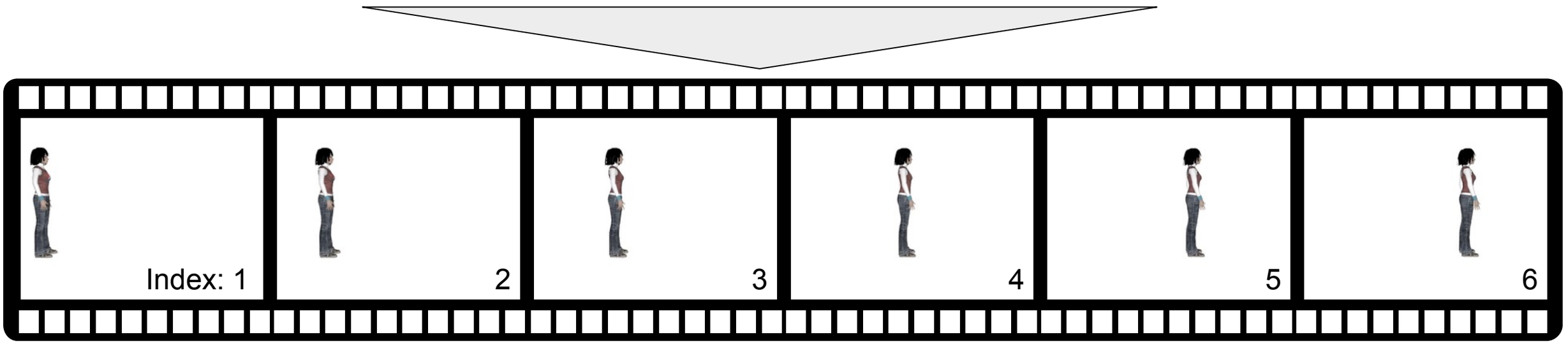
```
glm::mat4  lookAt (
detail::tvec3< T > const &eye,
detail::tvec3< T > const &center,
detail::tvec3< T > const &up)
```

Parameters to describe a camera position and orientation:

- eye: Specifies the position of the eye.
- center
  Specifies the position of the point to look to.
- up: Specifies the direction of the up vector.

Interactive Graphics Application

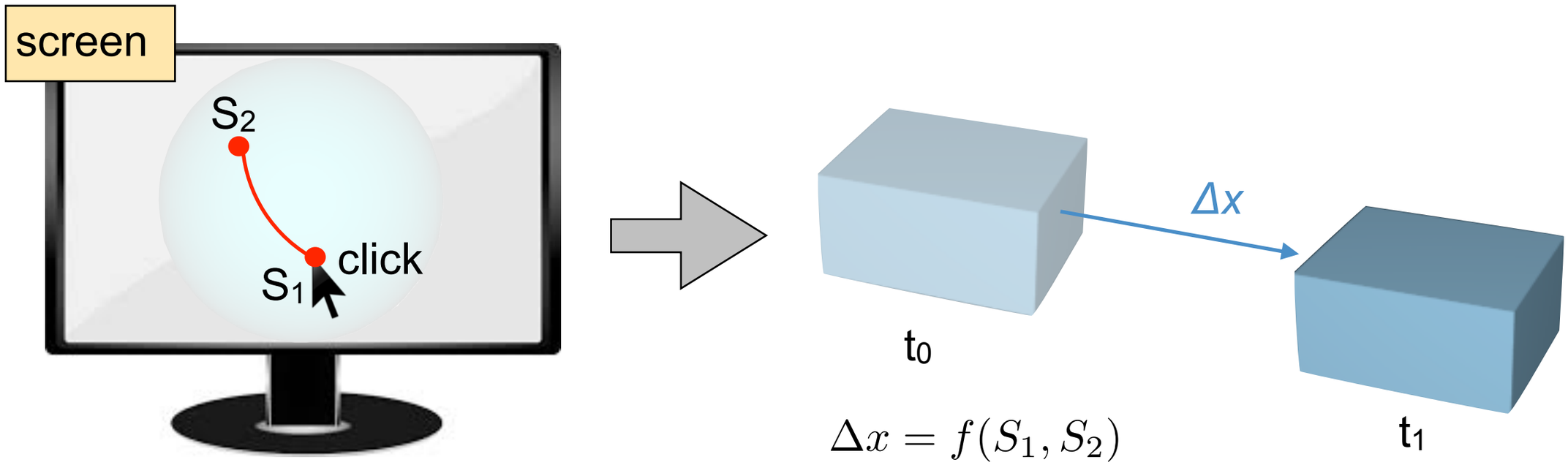




An interactive computer graphics application is generated as a sequence of successive frames. The frame rate is min 25Hz so that we do not notice single frames.

Relation input and action

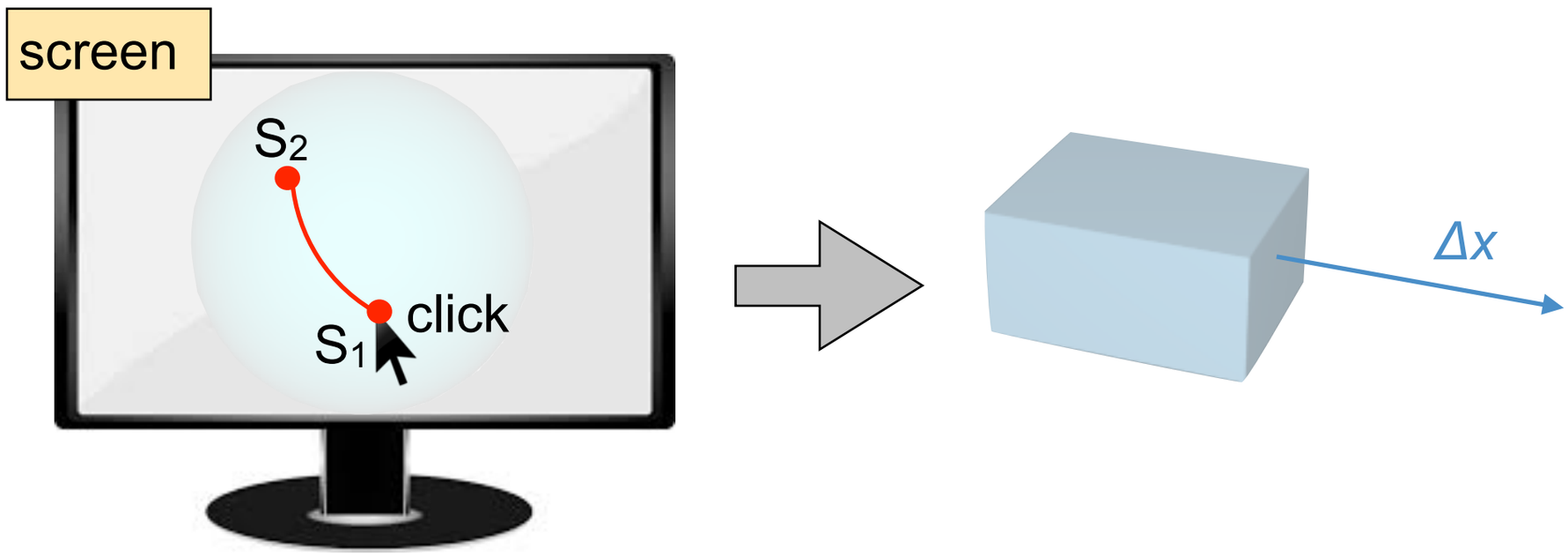
Direct action - direct reaction

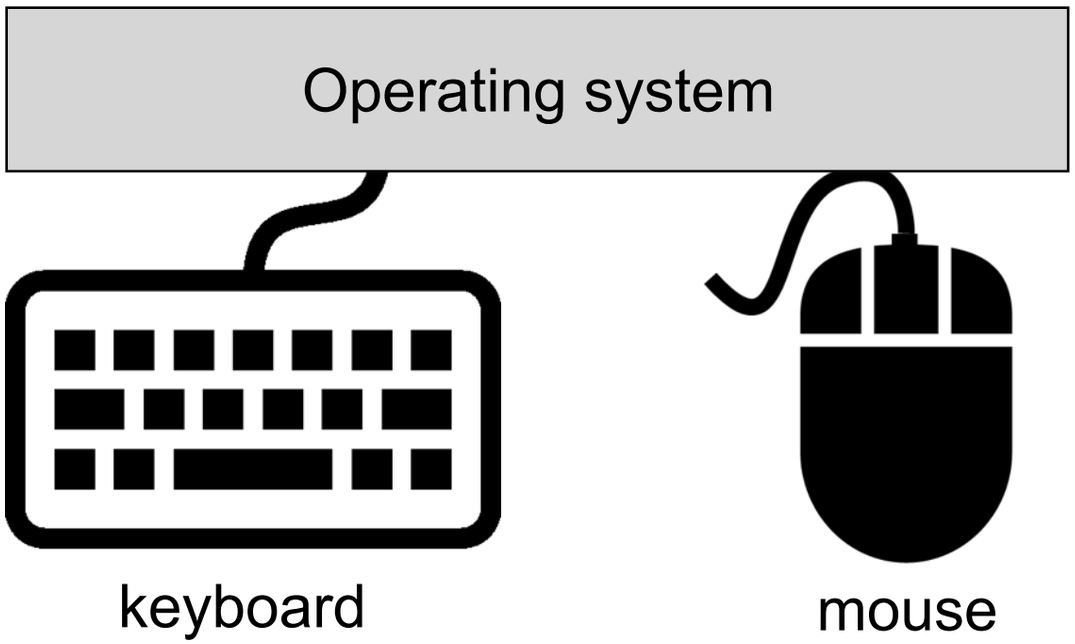


Motion on screen is directly transferred into a camera/object parameter.

- for instance, 5 units on screen = 5 "delta" steps in the virtual environment.
- Simple implementation
- But motion depends on the frame rate.

Relation input and action

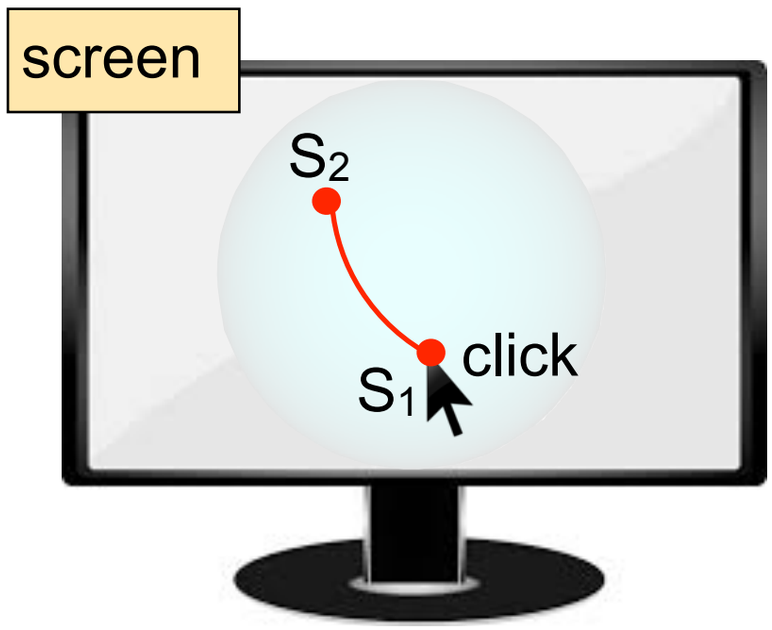


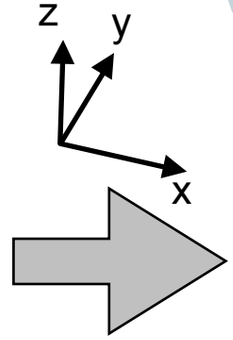


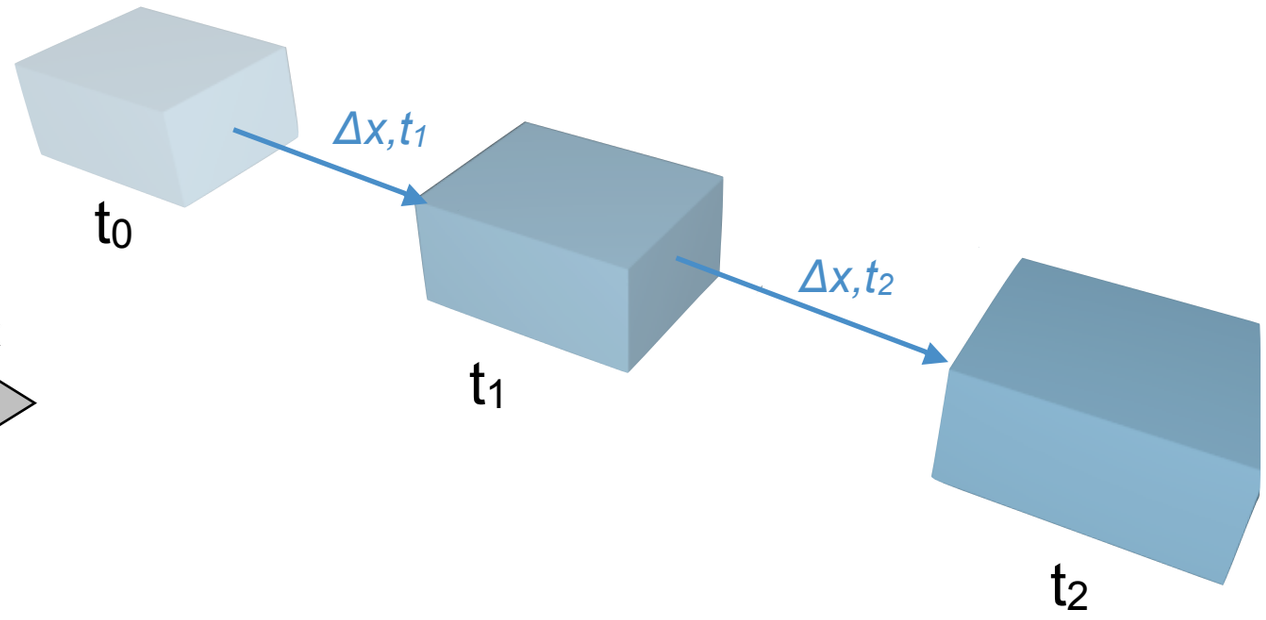
The mouse and keyboard (interaction devices in general) refresh rates does not necessarily operates in graphics frame rate. Especially k

Relation input and action

Direct action - indirect reaction



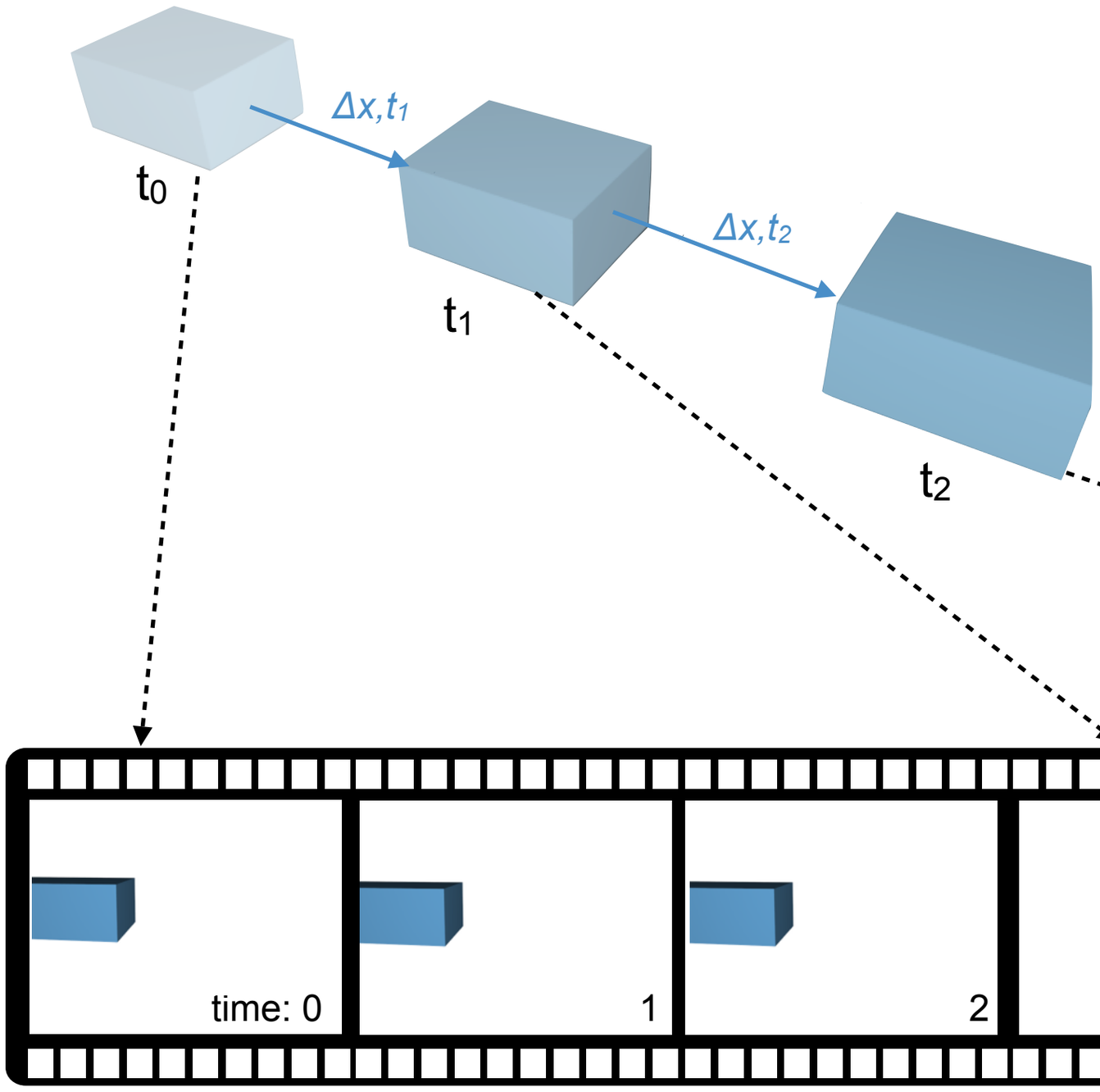




Motion on screen is transferred into a predefined distance on screen.

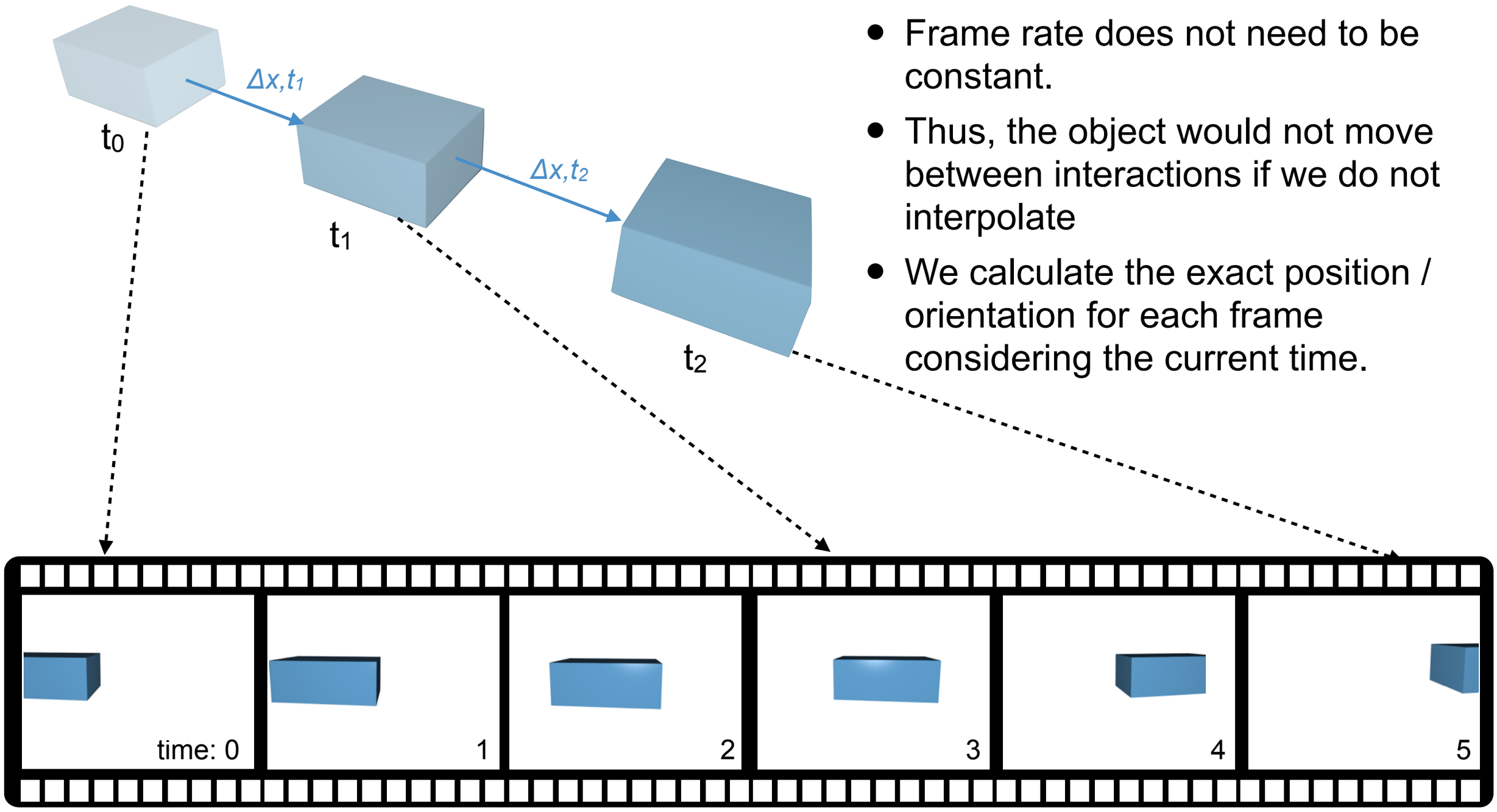
- 5 units on screen = 5 units - meters, inches, etc. - in the virtual environment.
- Requires interpolation since time delta is not always constant.
- Smooth motion.

NO Interpolation between Points



- Frame rate does not need to be constant.
- Thus, the object would not move between interactions if we do not interpolate
- We calculate the exact position / orientation for each frame considering the current time.

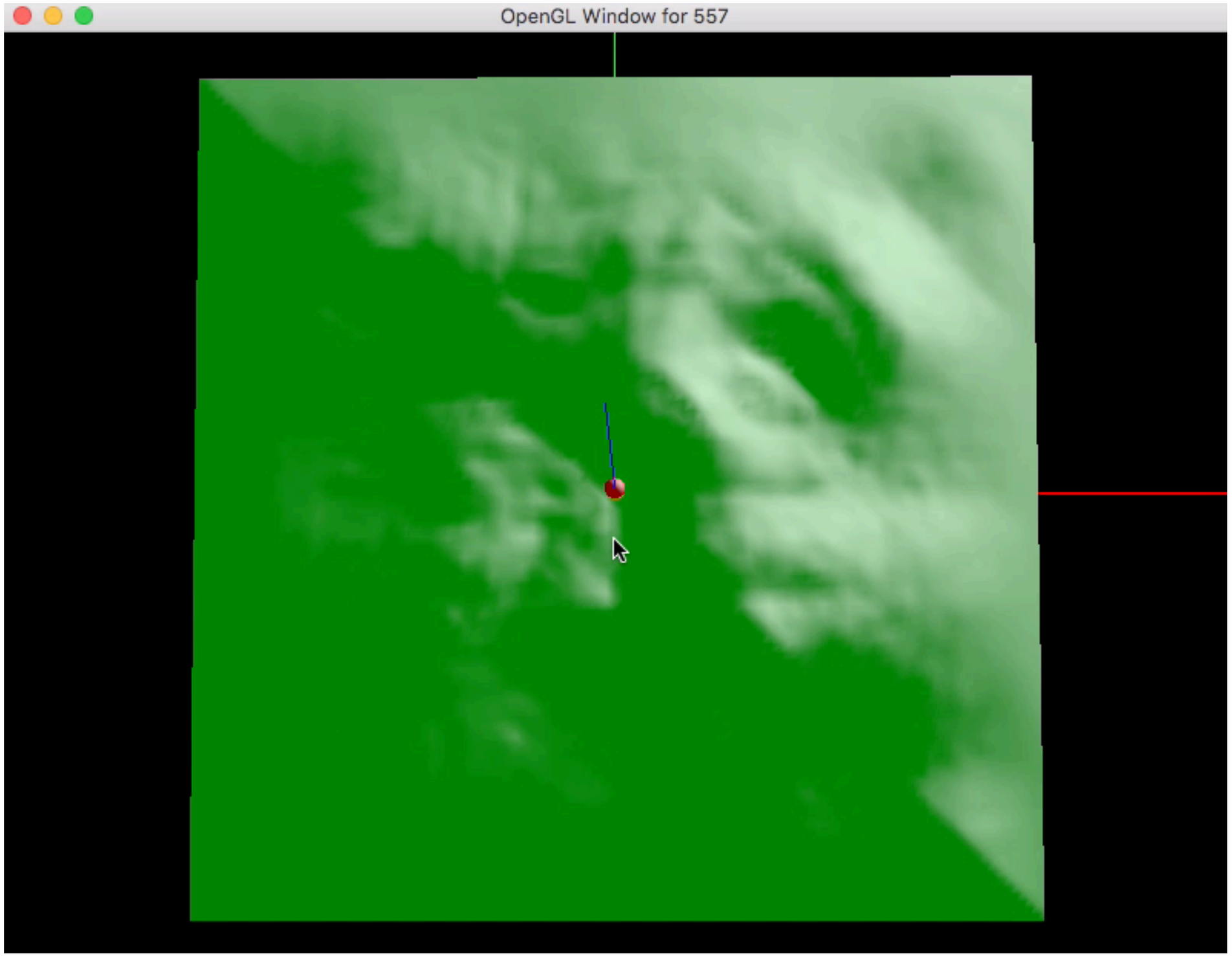
Interpolation between Points



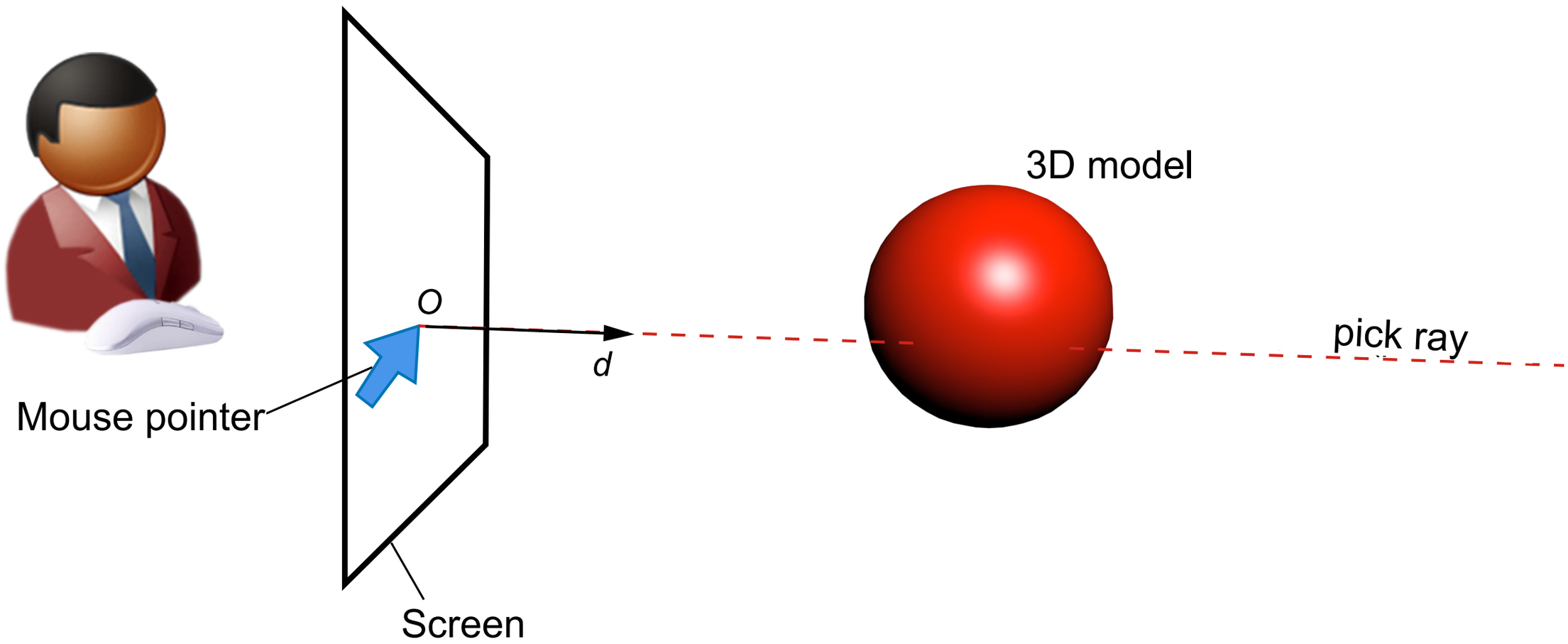
- Frame rate does not need to be constant.
- Thus, the object would not move between interactions if we do not interpolate
- We calculate the exact position / orientation for each frame considering the current time.

You do not want that!

ARLAB



Picking



Pick ray-method to select virtual objects in a 3D scene

In computer graphics, picking is implemented as ray-intersection test; collision detection between a ray and a 3D model. Therefore, a ray ***r(t)*** is sent into the 3D environment, it starts at ***O*** and is directed to ***d***.

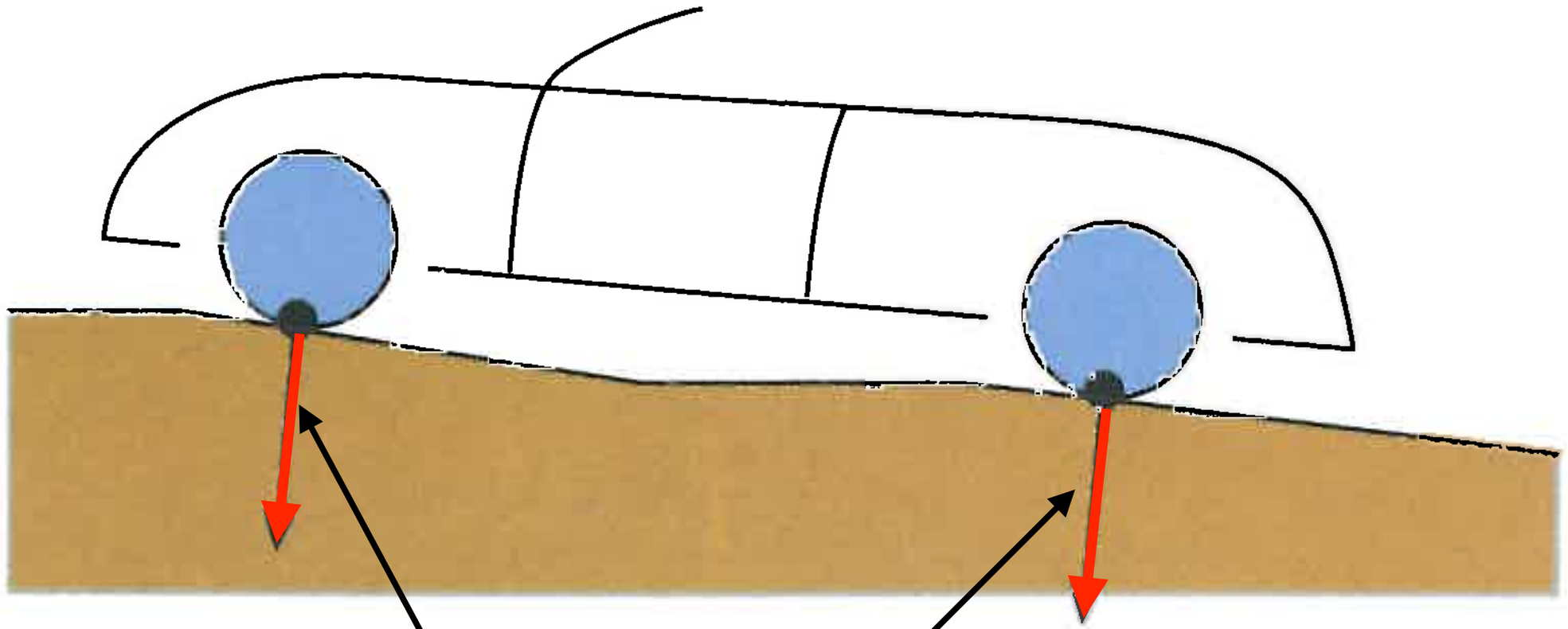
$$r(t) = O + td$$

with t, a scalar value to calculate intersection points along the ray.

If the ray “hits” a 3D object, this is considered as selection.

Locomotion and the Ray-Intersection-Test

ARLAB



We "shot" rays from a particular location to an object we are moving on. This allows us to identify the, e.g. ground plane and to move on this plane.

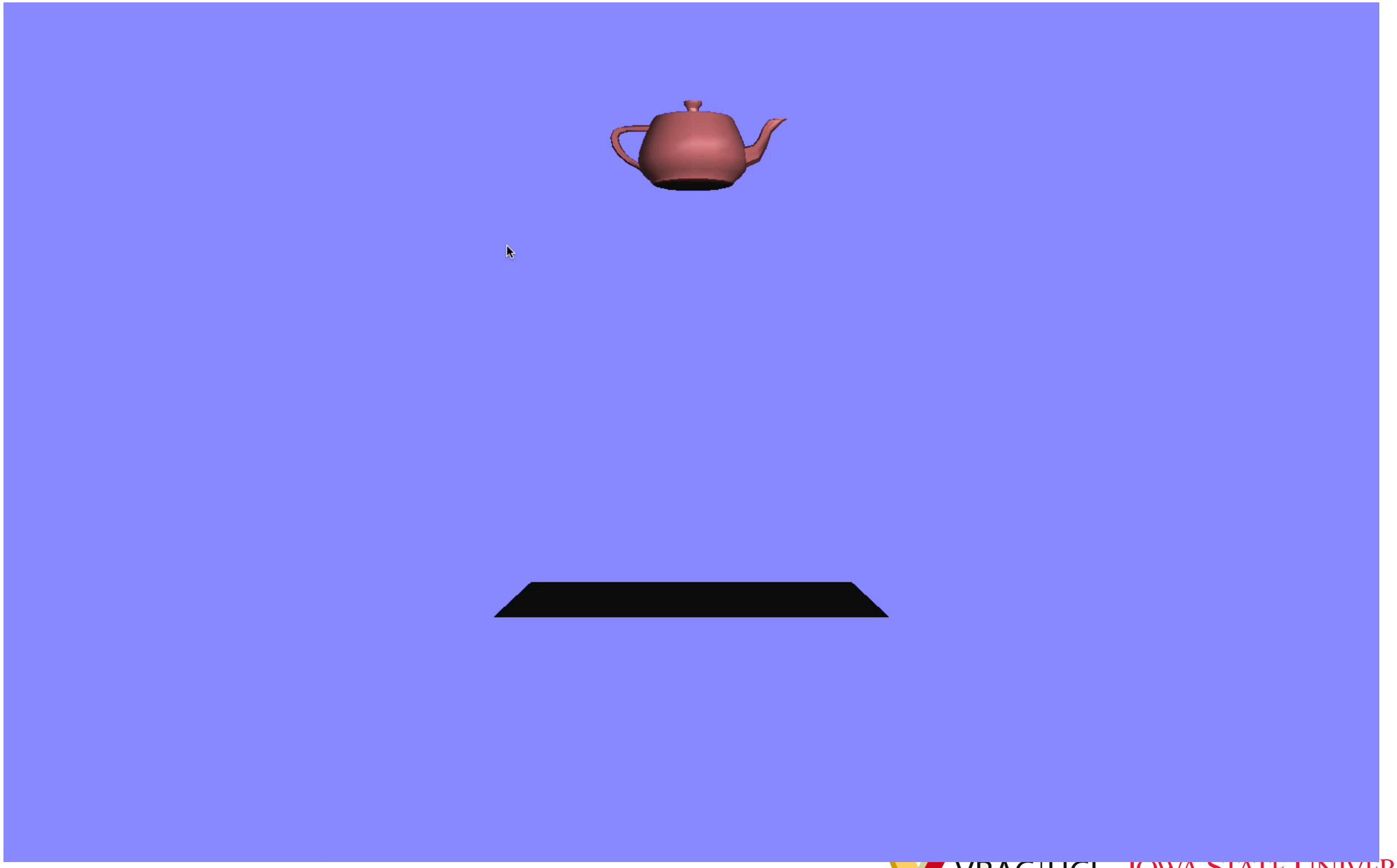
Quelle: Akenine-Möller et al.: Real-Time Rendering, 3rd Edition. A. K. Peters Ltd., Wellesley, MA, USA, 2008

VRAC|HCI

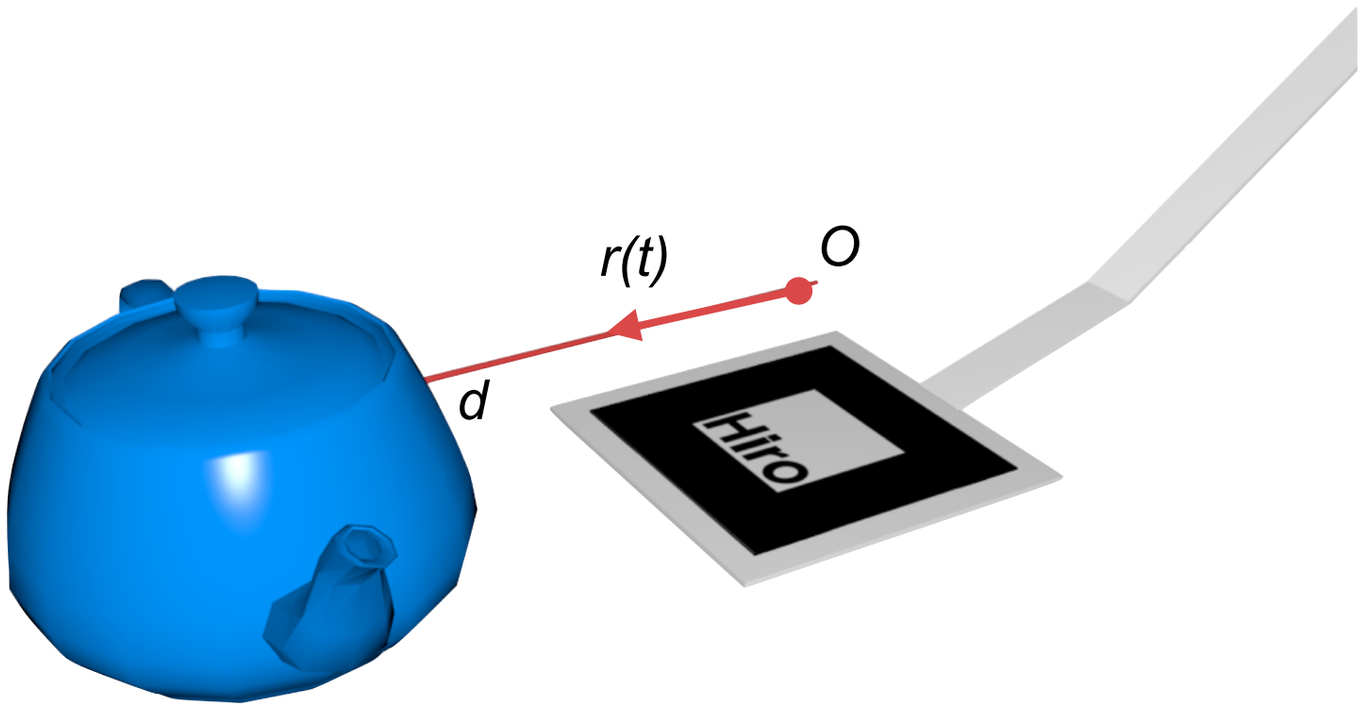
IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

video

ARLAB



Select



Select: a pick ray on an AR marker

We need a

- line / ray for intersection test
- code that computes the intersection between the ray and the objects in the scene.

The select operation utilizes a pick ray on an AR pattern. A ray-intersection test is carried out to detect 3D objects in front of that particular AR Paddle marker.

The ray is defined as:

$$r(t) = O + td$$

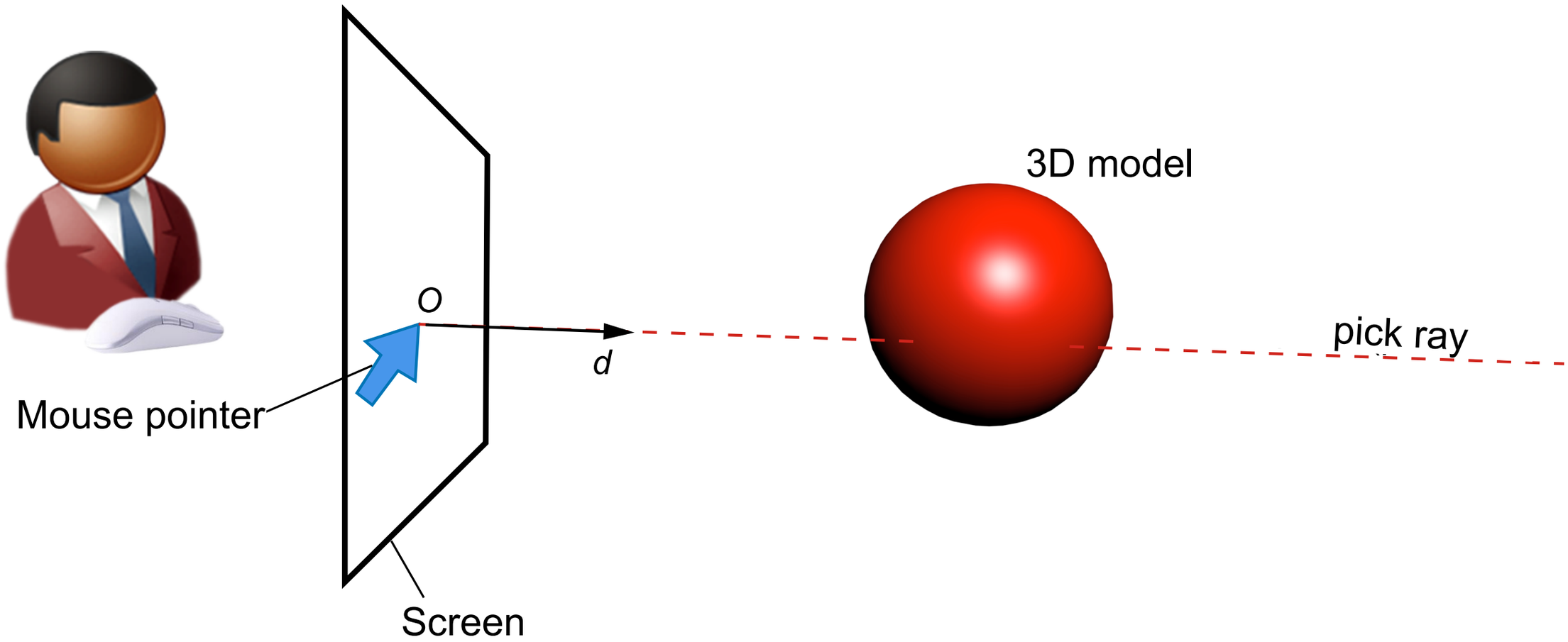
with *O*, the start point
d, the ray direction, and *t*, a unit vector.

The state machine for a selection test distinguish the states collision and no-collision.

An off-state is helpful, if the collision detection should be temporary suspended.

Ray-Intersection-Test

Ray-Intersection-Test



Pick ray-method to select virtual objects in a 3D scene

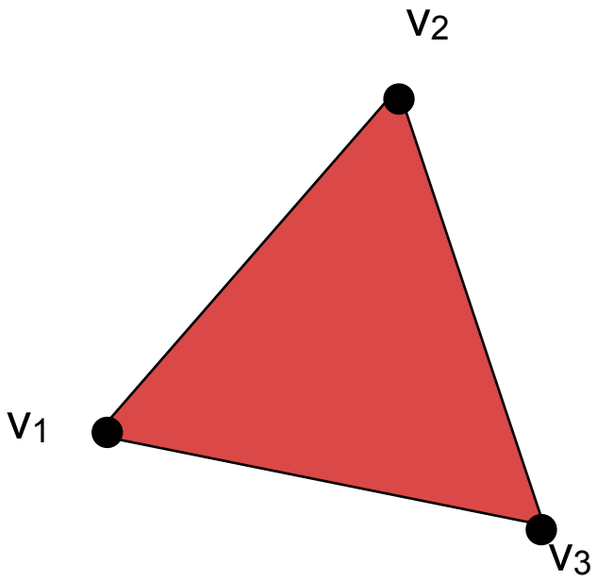
In computer graphics, picking is implemented as ray-intersection test; collision detection between a ray and a 3D model. Therefore, a ray ***r(t)*** is sent into the 3D environment, it starts at ***O*** and is directed to ***d***.

$$r(t) = O + td$$

with t, a scalar value to calculate intersection points along the ray.

If the ray “hits” a 3D object, this is considered as selection.

3D Objects

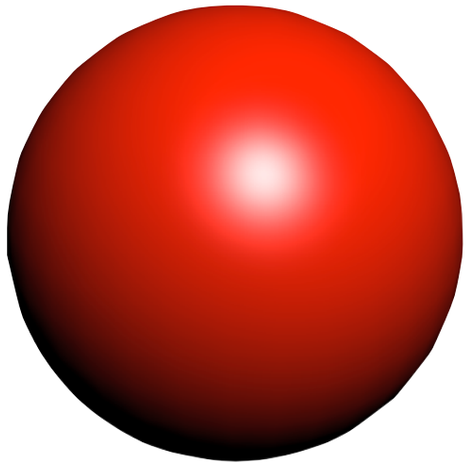


$$\mathbf{t}(u,v) = (1-u-v)\mathbf{v_0} + u\mathbf{v_1} + v\mathbf{v_2}$$

with $u \geq 0, v \geq 0$

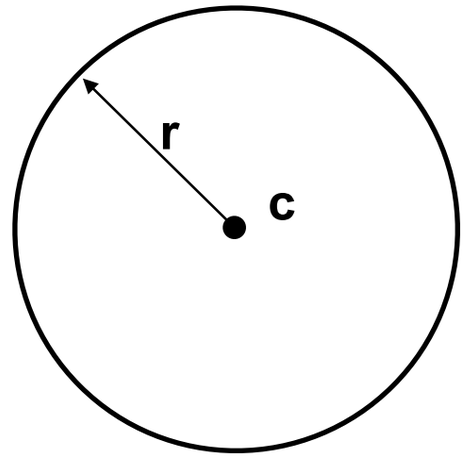
$u + v \leq 1$

Explicit representation of a triangle



$$f(\rho,\phi) = \begin{pmatrix} r\,sin(\rho)\,cos(\tau) \\ r\,sin(\rho)\,sin(\tau) \\ r\,cos(\tau) \end{pmatrix}$$

Explicit representation of a sphere



$$f(\mathbf{p}) = ||\mathbf{p} - \mathbf{c}|| - r = 0$$

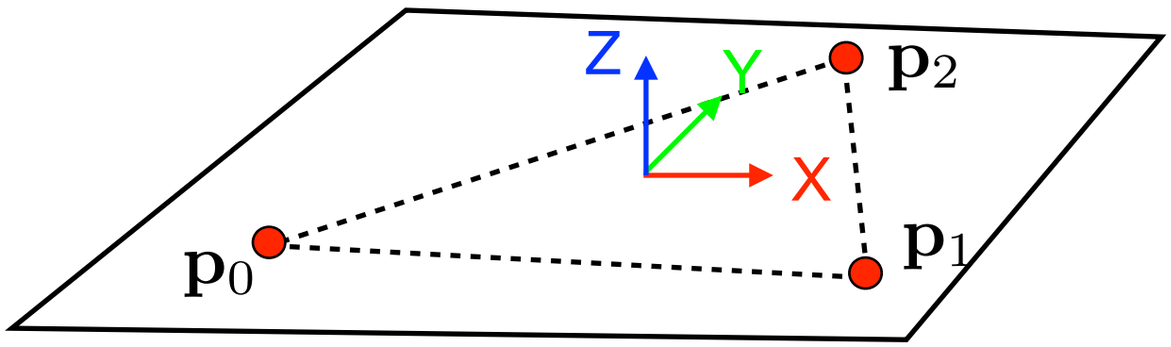
Simplified representation of a circle / sphere

In order to apply an intersection test, an explicit mathematical representation of an 3D object as vector function is required. The vector function must describe every surface point on the shell of the 3D model.

Line-Plane-Intersection

We can describe every plane with three points

$$\mathbf{p}_0 + (\mathbf{p}_1 - \mathbf{p}_0)u + (\mathbf{p}_2 - \mathbf{p}_0)v$$



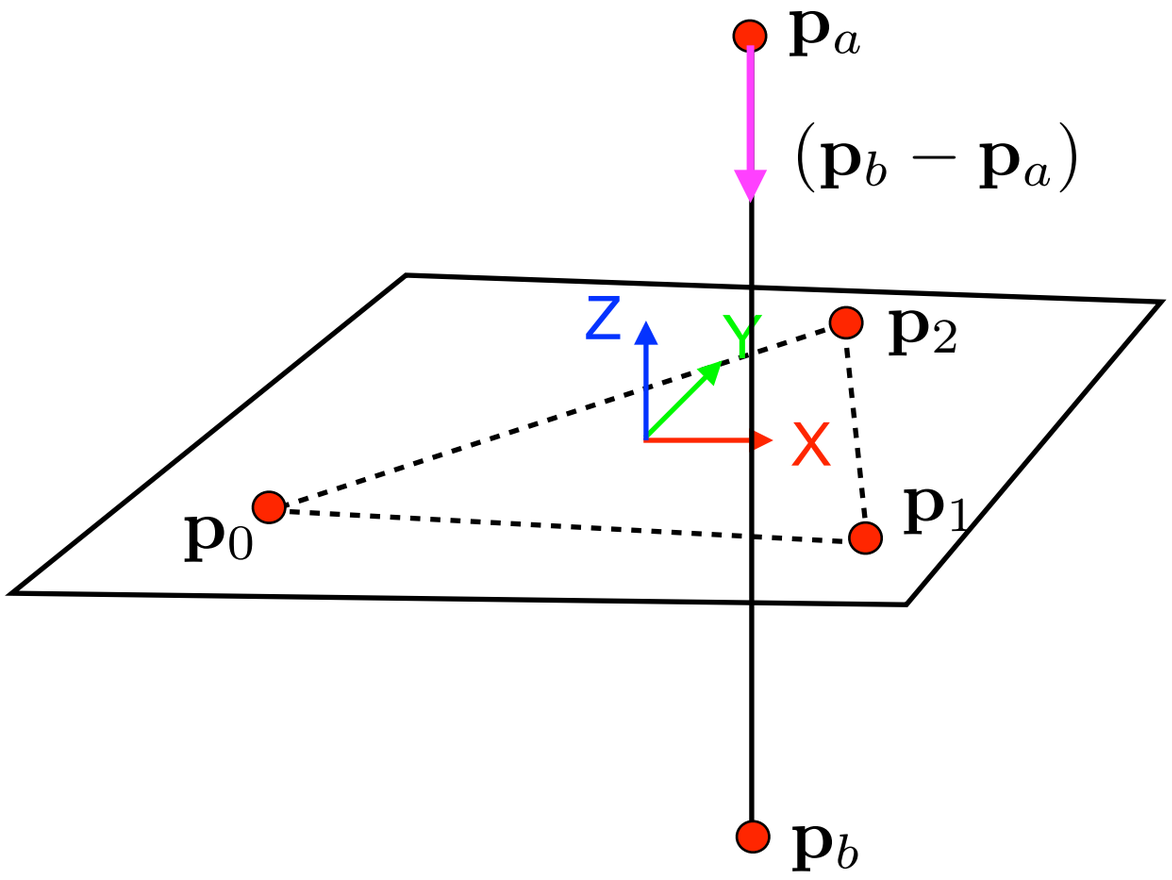
Line-Plane-Intersection

We can describe every plane with three points

$$\mathbf{p}_0 + (\mathbf{p}_1 - \mathbf{p}_0)u + (\mathbf{p}_2 - \mathbf{p}_0)v$$

A line can be represented by its start and endpoint

$$\mathbf{p}_a + (\mathbf{p}_b - \mathbf{p}_a)t$$



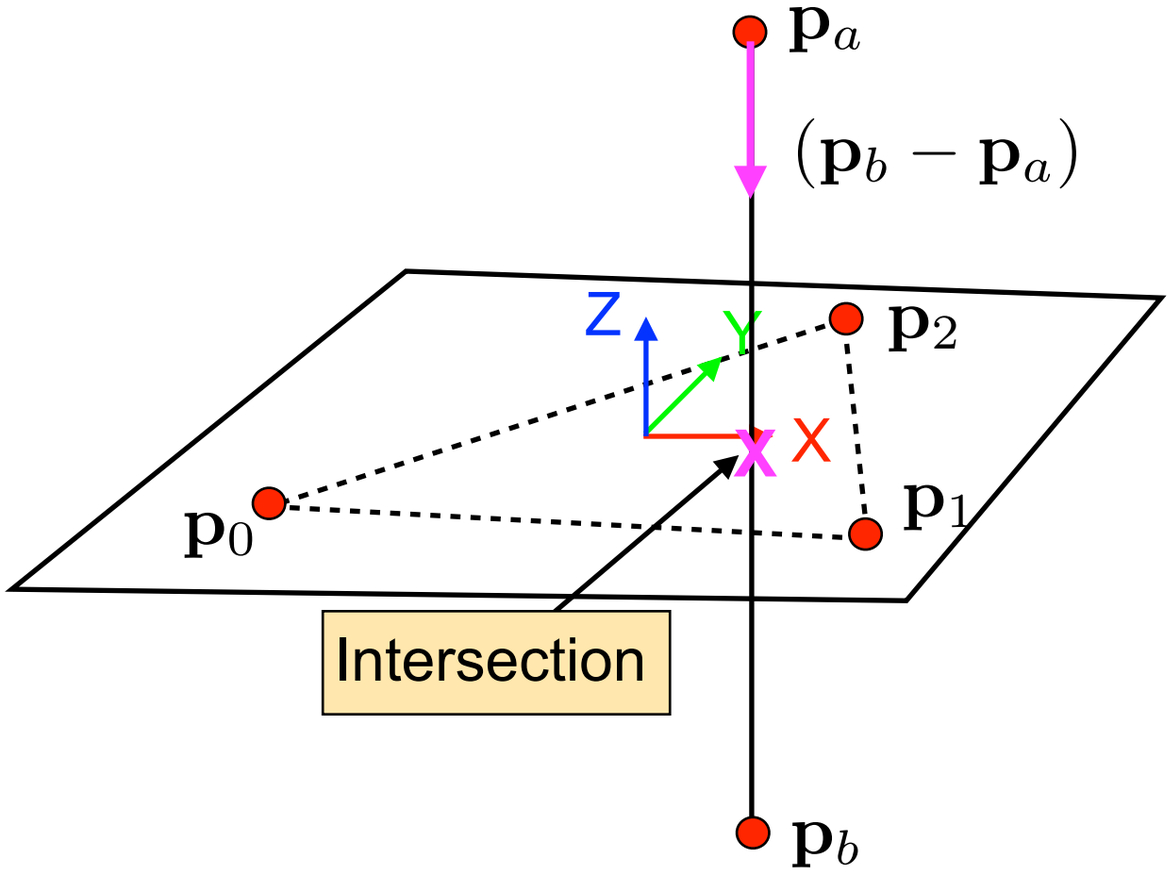
Line-Plane-Intersection

We can describe every plane with three points

$$\mathbf{p}_0 + (\mathbf{p}_1 - \mathbf{p}_0)u + (\mathbf{p}_2 - \mathbf{p}_0)v$$

A line can be represented by its start and endpoint

$$\mathbf{p}_a + (\mathbf{p}_b - \mathbf{p}_a)t$$



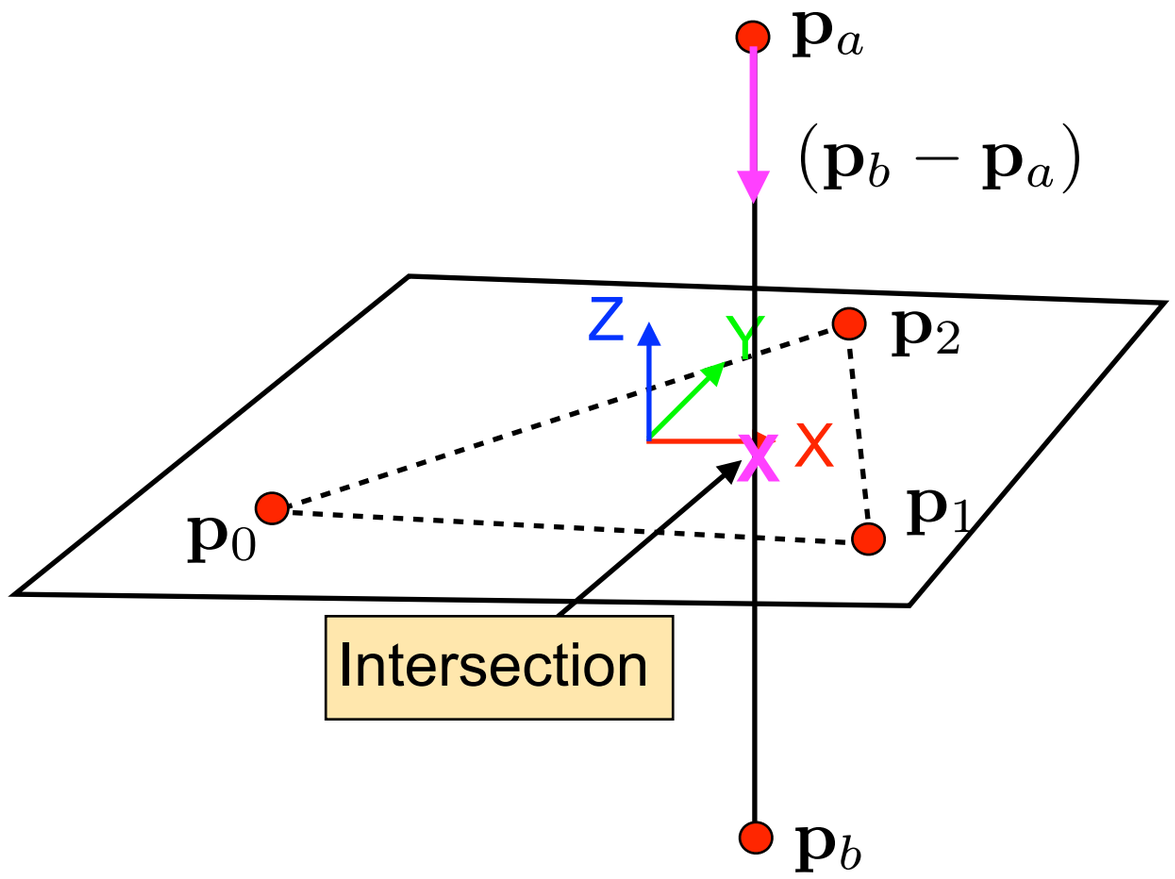
The point, at which the line intersects the plane is:

$$\mathbf{p}_0 + (\mathbf{p}_1 - \mathbf{p}_0)u + (\mathbf{p}_2 - \mathbf{p}_0)v = \mathbf{p}_a + (\mathbf{p}_b - \mathbf{p}_a)t$$

Line-Plane-Intersection

The point, at which the line intersects the plane is:

$$\mathbf{p}_0 + (\mathbf{p}_1 - \mathbf{p}_0)u + (\mathbf{p}_2 - \mathbf{p}_0)v = \mathbf{p}_a + (\mathbf{p}_b - \mathbf{p}_a)t$$



We can write this in matrix form

$$\begin{bmatrix} x_a - x_0 \\ y_a - y_0 \\ z_a - z_0 \end{bmatrix} = \begin{bmatrix} x_a - x_b & x_1 - x_0 & x_2 - x_0 \\ y_a - y_b & y_1 - y_0 & y_2 - y_0 \\ z_a - z_b & z_1 - z_0 & z_2 - z_0 \end{bmatrix} \begin{bmatrix} t \\ u \\ v \end{bmatrix}$$

$$\mathbf{l} = \mathbf{A}\mathbf{x}$$

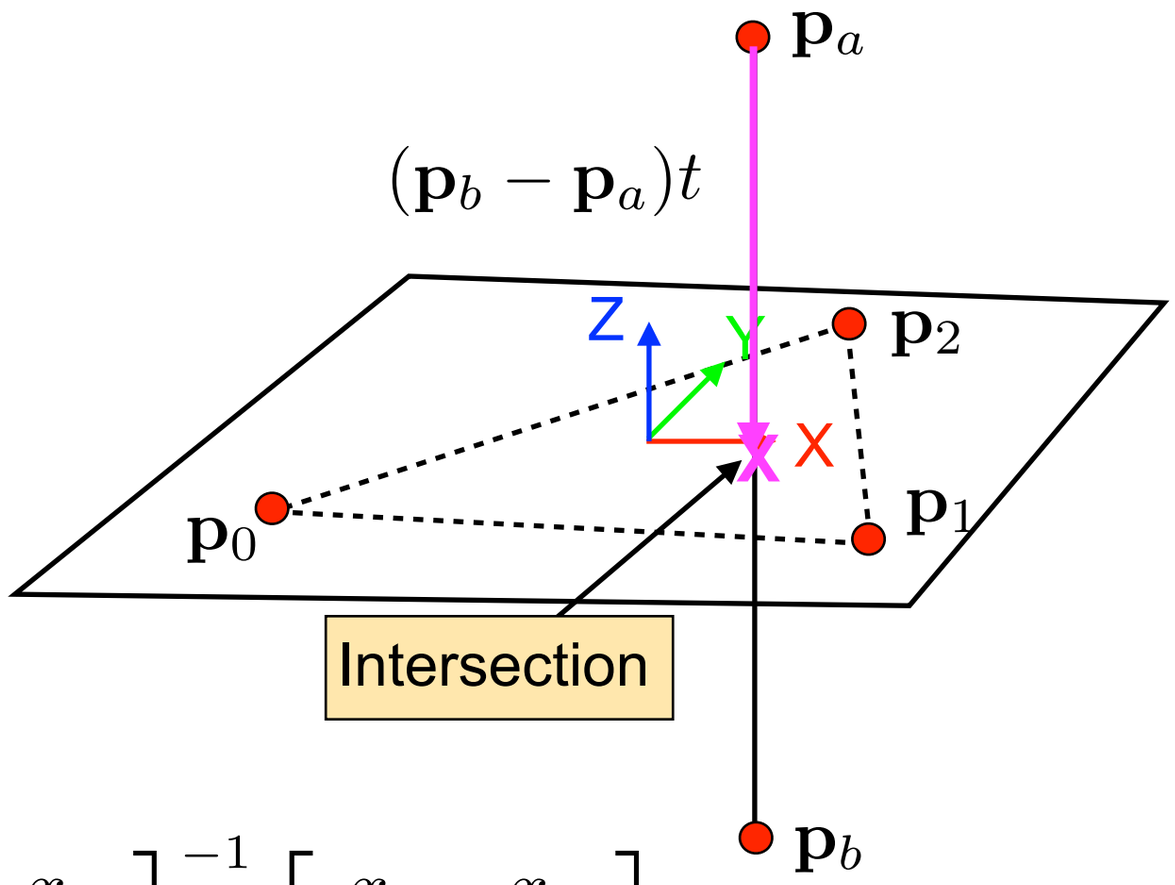
How do we solve this?

Line-Plane-Intersection

$$\mathbf{l} = \mathbf{A}\mathbf{x}$$

Solution:

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{l}$$



in matrix form:

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \begin{bmatrix} x_a - x_b & x_1 - x_0 & x_2 - x_0 \\ y_a - y_b & y_1 - y_0 & y_2 - y_0 \\ z_a - z_b & z_1 - z_0 & z_2 - z_0 \end{bmatrix}^{-1} \begin{bmatrix} x_a - x_0 \\ y_a - y_0 \\ z_a - z_0 \end{bmatrix}$$

The intersection is given as

$$\mathbf{p}_i = \mathbf{p}_a + (\mathbf{p}_b - \mathbf{p}_a)t$$

Line-Plane-Intersection Example

The line is given by:

$$\mathbf{p}_a = \{2, 0, 10\}$$

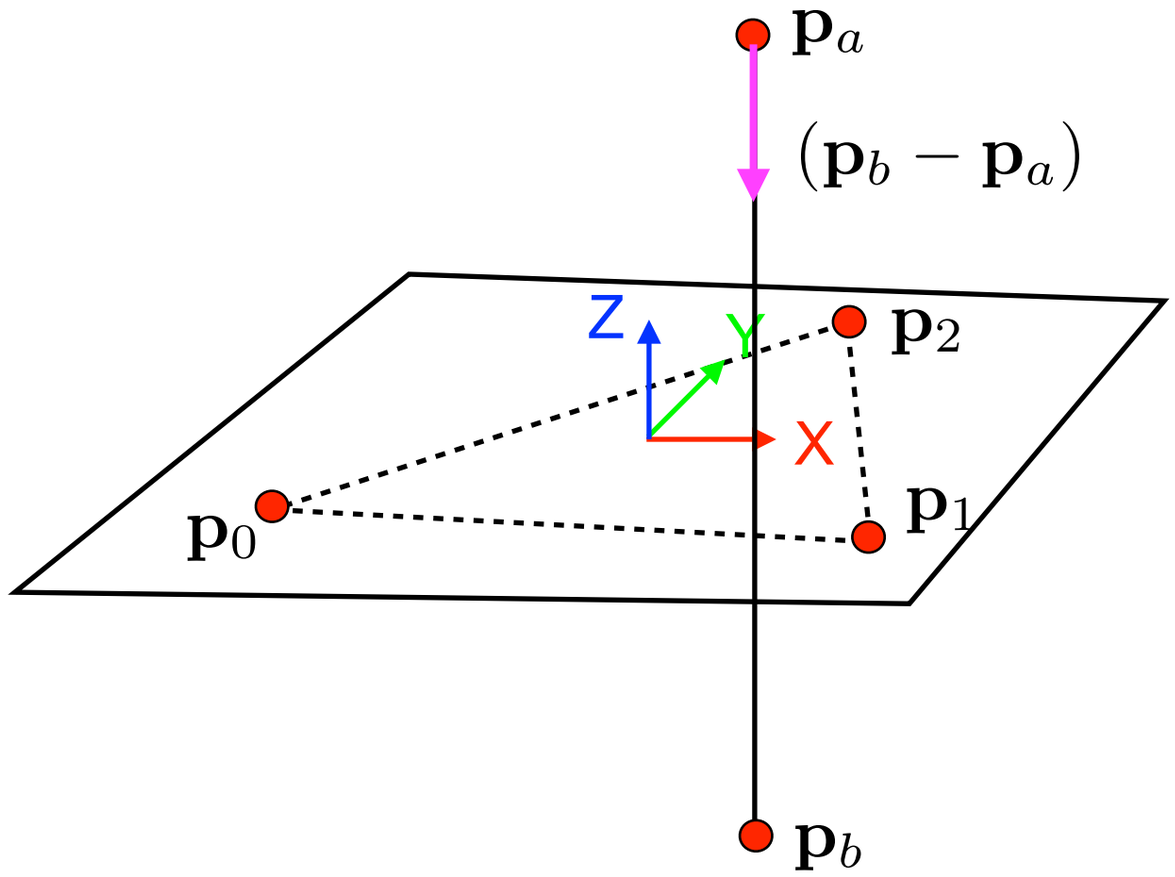
$$\mathbf{p}_b = \{2, 0, -10\}$$

The plane is given by:

$$\mathbf{p}_0 = \{-5, -5, 0\}$$

$$\mathbf{p}_1 = \{5, -5, 0\}$$

$$\mathbf{p}_2 = \{2, 5, 0\}$$



Our matrix is:

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \begin{bmatrix} 2-2 & 5-(-5) & 2-(-5) \\ 0-0 & -5-(-5) & 5-(-5) \\ 10-(-10) & 0-0 & 0-0 \end{bmatrix}^{-1} \begin{bmatrix} 2-(-5) \\ 0-(-5) \\ 10-0 \end{bmatrix}$$

Line-Plane-Intersection Example

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \begin{bmatrix} 2-2 & 5-(-5) & 2-(-5) \\ 0-0 & -5-(-5) & 5-(-5) \\ 10-(-10) & 0-0 & 0-0 \end{bmatrix}^{-1} \begin{bmatrix} 2-(-5) \\ 0-(-5) \\ 10-0 \end{bmatrix}$$

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \begin{bmatrix} 0 & 10 & 7 \\ 0 & 0 & 10 \\ 20 & 0 & 0 \end{bmatrix}^{-1} \begin{bmatrix} 7 \\ 5 \\ 10 \end{bmatrix}$$

Inverting the matrix:

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \begin{bmatrix} 0.0 & 0.0 & 0.05 \\ 0.1 & -0.07 & 0.0 \\ 0.0 & 0.1 & 0.0 \end{bmatrix} \begin{bmatrix} 7 \\ 5 \\ 10 \end{bmatrix}$$

we are only interested in *t*:

$$t = 0.05 \cdot 10.0 = 0.5$$

Line-Plane-Intersection Example

The line is given by:

$$\mathbf{p}_a = \{2, 0, 10\}$$

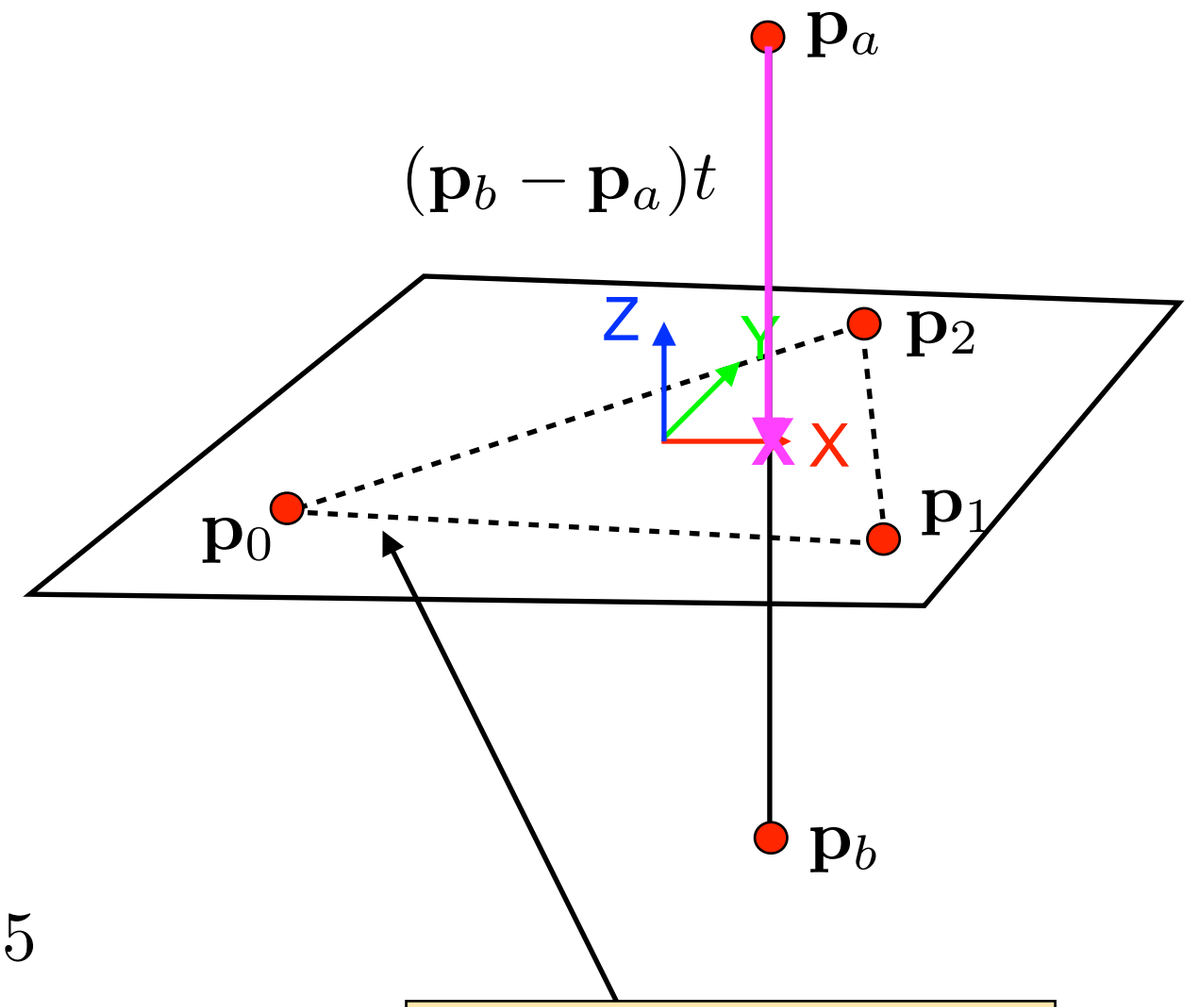
$$\mathbf{p}_b = \{2, 0, -10\}$$

$$\mathbf{p}_i = \mathbf{p}_a + (\mathbf{p}_b - \mathbf{p}_a)t$$

Result

$$\mathbf{p}_i = \begin{bmatrix} 2 \\ 0 \\ 10 \end{bmatrix} + \begin{bmatrix} 2-2 \\ 0-0 \\ -10-10 \end{bmatrix} \cdot 0.5$$

X $$\mathbf{p}_i = \begin{bmatrix} 2 \\ 0 \\ 10 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ -20 \cdot 0.5 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix}$$



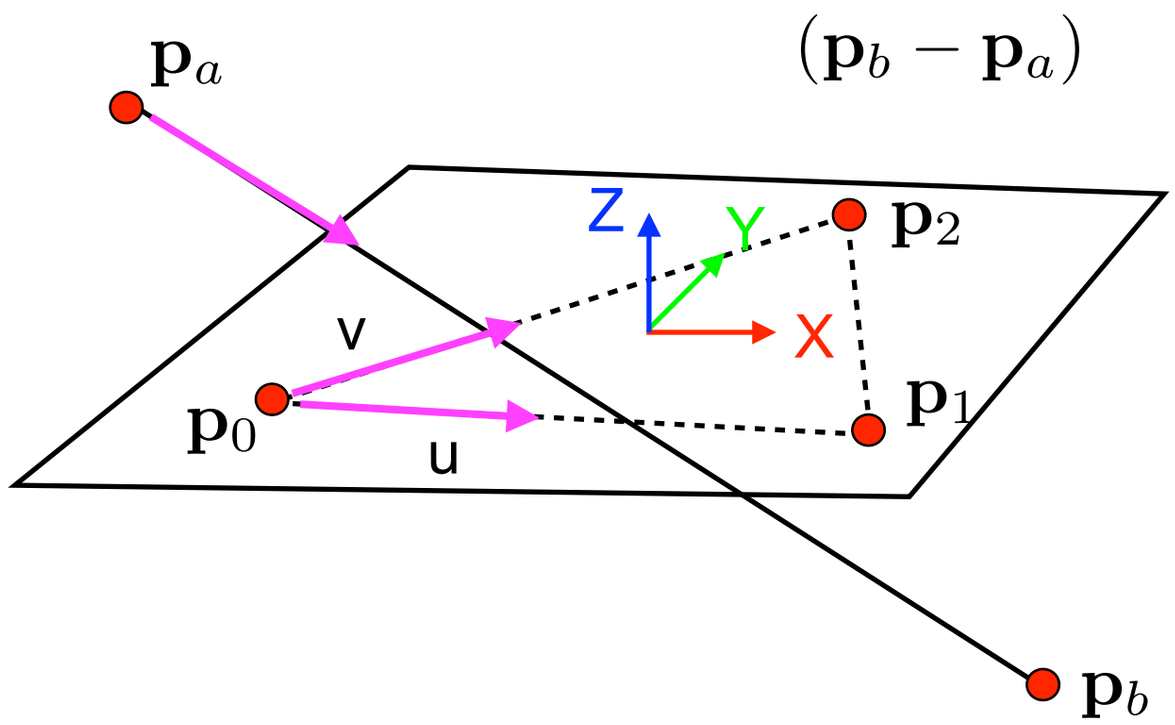
Every plane can be represented with three points = **triangle**

Line-Plane-Intersection

We do not have a solution when the line lies in the plane or runs parallel to the plane. Matrix **A** is singular in this case, determinant = 0

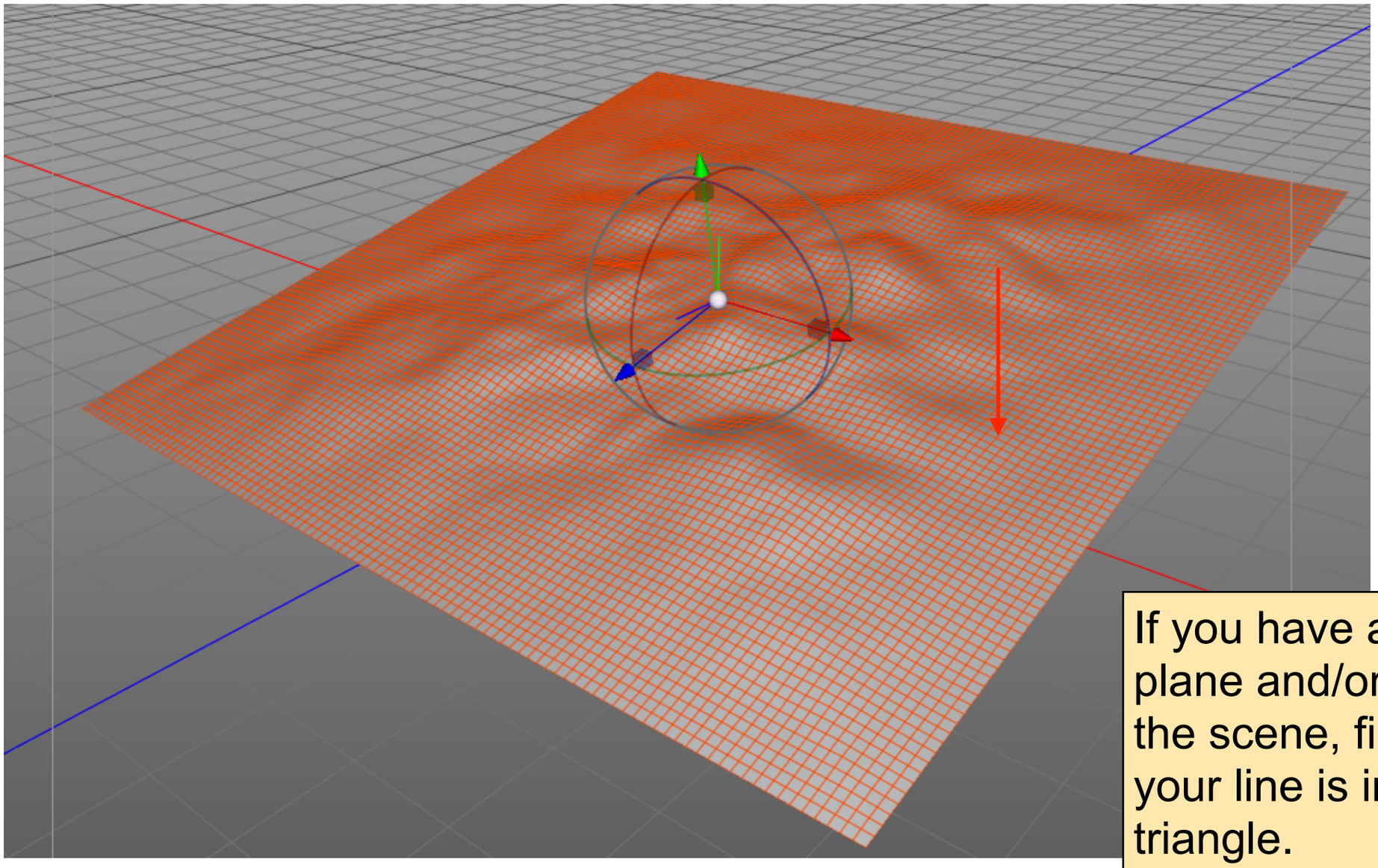
The parameters *u* and *v* run in an interval *u* = [0,1] , *v* = [0. 1]. Otherwise, the line is not within the triangle.

The parameter t runs in the interval [0,1]. Otherwise the intersection is not within the line segment.



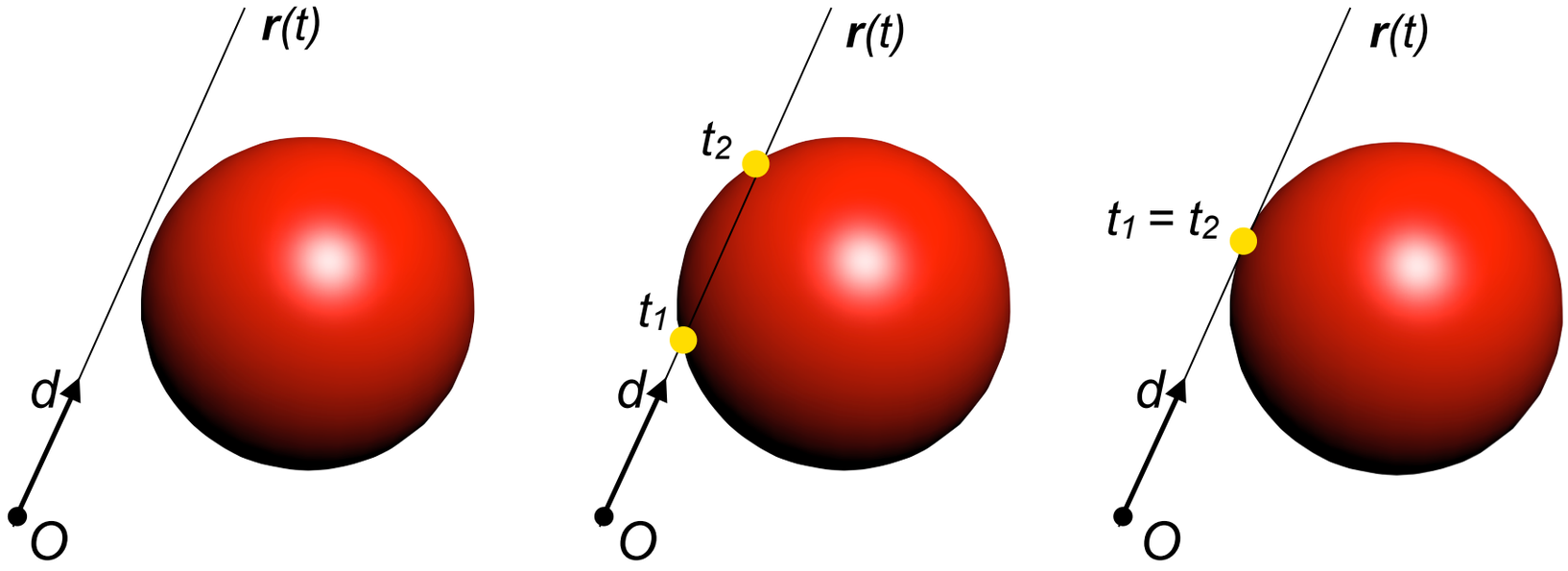
Practical Issues

Check distance and normal vectors first.



If you have a huge ground plane and/or multiple objects in the scene, first check whether your line is in vicinity to the triangle.

Ray-Intersection Test



Case 1: Complex solution - the ray does not hit the 3D object.
Case 2: Two real solutions - the ray intersects with the sphere at point t_1 and t_2
Case 3: Two identical real solutions: the ray intersects with the sphere at one surface point.

Different cases of a ray-intersection text

Representation of a sphere $f(\mathbf{p}) = ||\mathbf{p} - \mathbf{c}|| - r = 0$

To solve the equation, include ***r(t)*** $f(\mathbf{r}(t)) = ||\mathbf{r}(t) - \mathbf{c}|| - r = 0$

with $r(t) = O + td$

and $\mathbf{d} \cdot \mathbf{d} = ||\mathbf{d}||^2 = 1$

Solution $\boxed{t^2 + 2tb + c = 0}$

with $b = \mathbf{d} \cdot (O - \mathbf{c})$

and $c = (O - \mathbf{c}) \cdot (O - \mathbf{c}) - r^2$

ARLAB

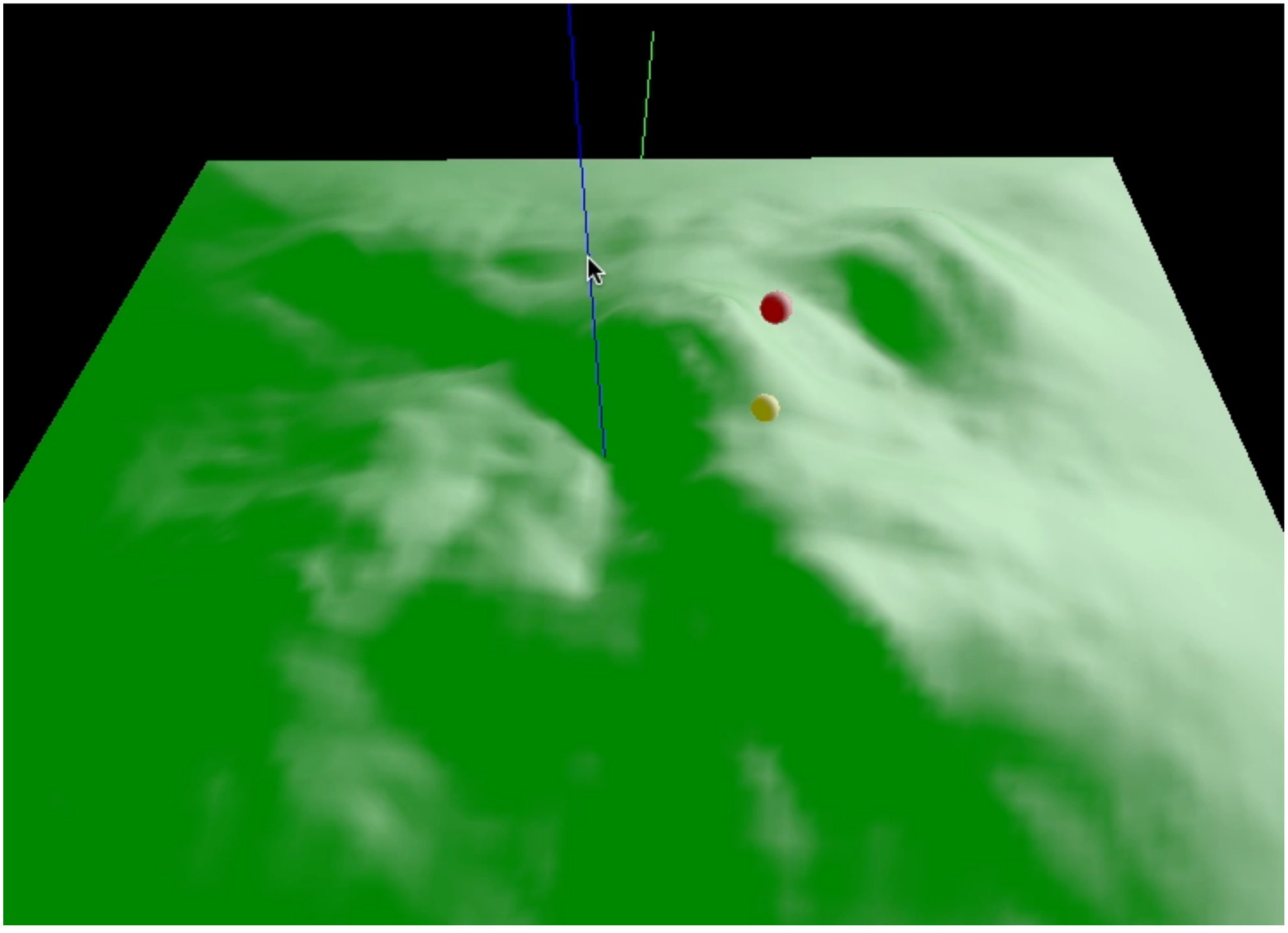
Code Example

VRAC|HCI

IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

GLM matrix inversion

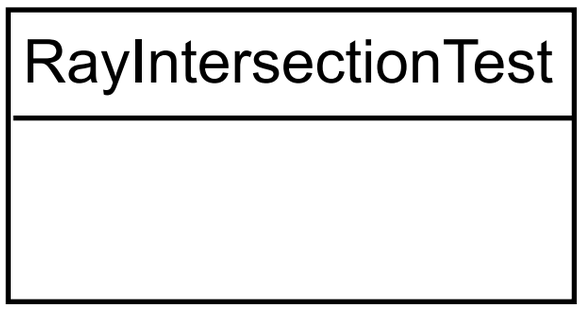
ARLAB



VRAC|HCI

IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

RayIntersectionTest



The class RayIntersectionTest implements a line/plane intersection

Static member:

```
static bool intersect(   const glm::vec3& ray_start,
                         const glm::vec3&  ray_stop,
                         GLObjectObj& object,
                         vector<glm::vec3>& intersect_list);
```

Parameters

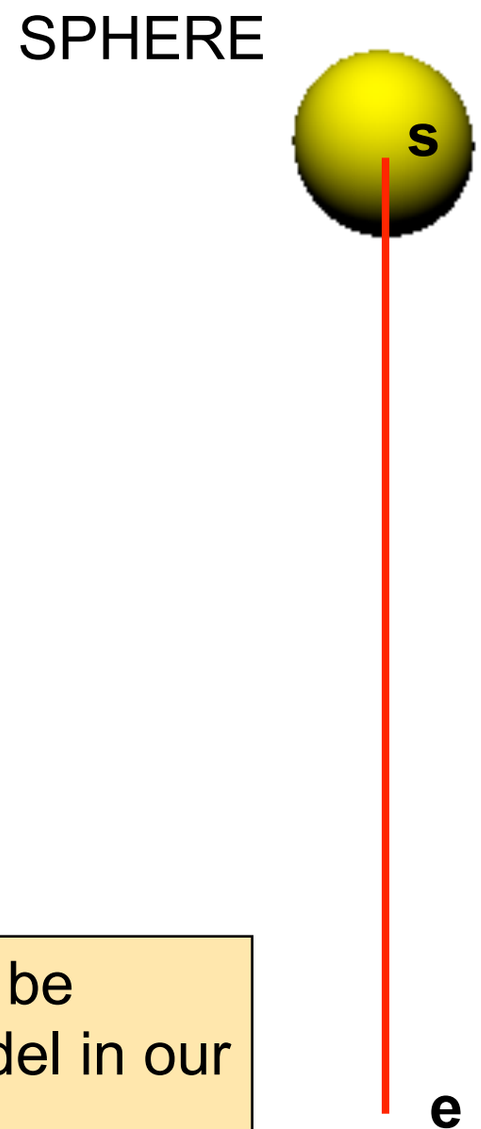
- ray_start: start point of the ray in world coordinates
- ray_stop: end point of the ray in world coordinates
- object: reference to the object in world coordinates
- intersect_list: vector with all intersection results as glm::vec3

RayIntersectionTest

Example:

```
// to transform the sphere
glm::mat4 g_tranform_sphere;

  [...]

glm::vec3 s(  g_tranform_sphere[3][0],
              g_tranform_sphere[3][1],
              g_tranform_sphere[3][2]);
glm::vec3 e(  g_tranform_sphere[3][0],
              g_tranform_sphere[3][1],
              g_tranform_sphere[3][2]-20);

vector<glm::vec3> res;

// perform the ray intersectiont test.
RayIntersectionTest::intersect(s, e, *loadedModel1, res);
```

The model that should be tested. The terrain model in our example.



intersect(.....)

ARLAB

```
bool RayIntersectionTest::intersect(const glm::vec3& ray_start, const glm::vec3&  ray_stop, GLObjectObj& object,
vector<glm::vec3>& intersect_list)
{
//#define TEST
#ifdef TEST
   [...]
#else
    // get the current model matrix.
    glm::mat4 m = object._modelMatrix;
    glm::mat4 mInv = glm::inverse(m);

    // get all verticees
    vector<glm::vec3> vertices = object._vertices;

    glm::vec3 s = ray_start;
    glm::vec3 e = ray_stop;
#endif
    // for all vertices
    vector<glm::vec3>::iterator itr = vertices.begin();
    while (itr != vertices.end())
    {
        // get one vertex
        glm::vec3 p0 = (*itr);itr++;
        glm::vec3 p1 = (*itr);itr++;
        glm::vec3 p2 = (*itr);itr++;


        glm::vec4 v0 = m * glm::vec4(p0.x, p0.y, p0.z, 1.0);
        glm::vec4 v1 = m * glm::vec4(p1.x, p1.y, p1.z, 1.0);
        glm::vec4 v2 = m * glm::vec4(p2.x, p2.y, p2.z, 1.0);

        p0.x = v0.x; p0.y = v0.y; p0.z = v0.z;
        p1.x = v1.x; p1.y = v1.y; p1.z = v1.z;
        p2.x = v2.x; p2.y = v2.y; p2.z = v2.z;
```

intersect(.....)

The first lines just rewrite the variables into local variables.

```
// get the current model matrix.
glm::mat4 m = object._modelMatrix;
glm::mat4 mInv = glm::inverse(m);

// get all verticees
vector<glm::vec3> vertices = object._vertices;
```

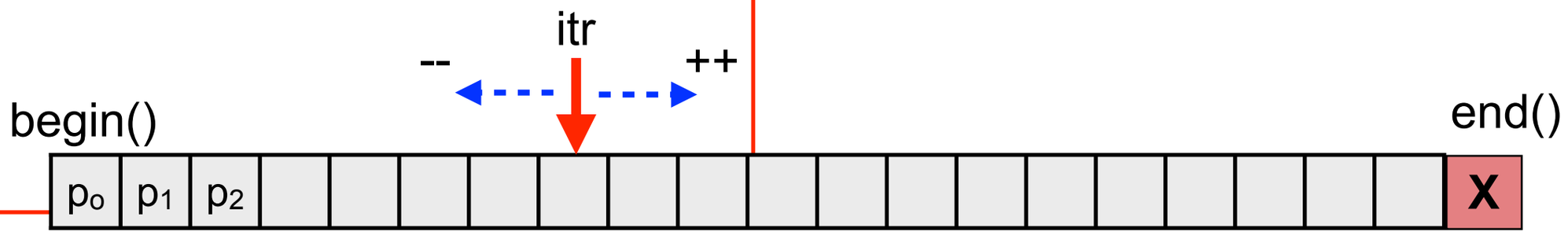
```
glm::vec3 s = ray_start;
glm::vec3 e = ray_stop;
```

intersect(.....)

```
// for all vertices

vector<glm::vec3>::iterator itr = vertices.begin();

while (itr != vertices.end())
{

      // get one vertex
      glm::vec3 p0 = (*itr);itr++;
      glm::vec3 p1 = (*itr);itr++;
      glm::vec3 p2 = (*itr);itr++;



}
```

The calculations are executed in a while-loop. We run through an iterator, fetch three points - we expect that three points are one triangle, and calculate the intersection.

[Intersection calculation]



intersect(.....)

```
// multiply the vertex with the model matrix
glm::vec4 v0 = m * glm::vec4(p0.x, p0.y, p0.z, 1.0);
glm::vec4 v1 = m * glm::vec4(p1.x, p1.y, p1.z, 1.0);
glm::vec4 v2 = m * glm::vec4(p2.x, p2.y, p2.z, 1.0);

// copy the values back into a vector 3
p0.x = v0.x; p0.y = v0.y; p0.z = v0.z;
p1.x = v1.x; p1.y = v1.y; p1.z = v1.z;
p2.x = v2.x; p2.y = v2.y; p2.z = v2.z;
```

```
// prepare the matrix a;
glm::mat3 A = glm::mat3();
A[0][0] = s.x - e.x;  A[0][1] = p1.x - p0.x;  A[0][2] = p2.x - p0.x;
A[1][0] = s.y - e.y;  A[1][1] = p1.y - p0.y;  A[1][2] = p2.y - p0.y;
A[2][0] = s.z - e.z;  A[2][1] = p1.z - p0.z;  A[2][2] = p2.z - p0.z;
```

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \begin{bmatrix} x_a - x_b & x_1 - x_0 & x_2 - x_0 \\ y_a - y_b & y_1 - y_0 & y_2 - y_0 \\ z_a - z_b & z_1 - z_0 & z_2 - z_0 \end{bmatrix}^{-1} \begin{bmatrix} x_a - x_0 \\ y_a - y_0 \\ z_a - z_0 \end{bmatrix}$$

intersect(.....)

ARLAB

```
// invert A;
glm::mat3 invA = glm::inverse(A);
```

Calculate the inverse

```
// prepare the vector [t,u,v]
glm::vec3 tuv;
tuv.x = s.x - p0.x;
tuv.y = s.y - p0.y;
tuv.z = s.z - p0.z;
```

```
// multiply
glm::vec3 result = glm::transpose(invA) * tuv;


float t = result.x;
float u = result.y;
float v = result.z;
```

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \begin{bmatrix} x_a - x_b & x_1 - x_0 & x_2 - x_0 \\ y_a - y_b & y_1 - y_0 & y_2 - y_0 \\ z_a - z_b & z_1 - z_0 & z_2 - z_0 \end{bmatrix}^{-1} \begin{bmatrix} x_a - x_0 \\ y_a - y_0 \\ z_a - z_0 \end{bmatrix}$$

intersect(.....)

```
 // Check if we are still within the triangle.
if(u>=0.0 && u<= 1.0 && v >=0.0 & v<=1.0)
{
    float uv = u + v;
#ifdef TEST
  [....]
#endif

    if(uv >= 0.0 && uv <= 1.0  && t >= 0.0 && t <= 1.0)
    {

        // intersetcion
        glm::vec3 x = s + (e - s)*t;

        intersect_list.push_back(x);
    }
}
```

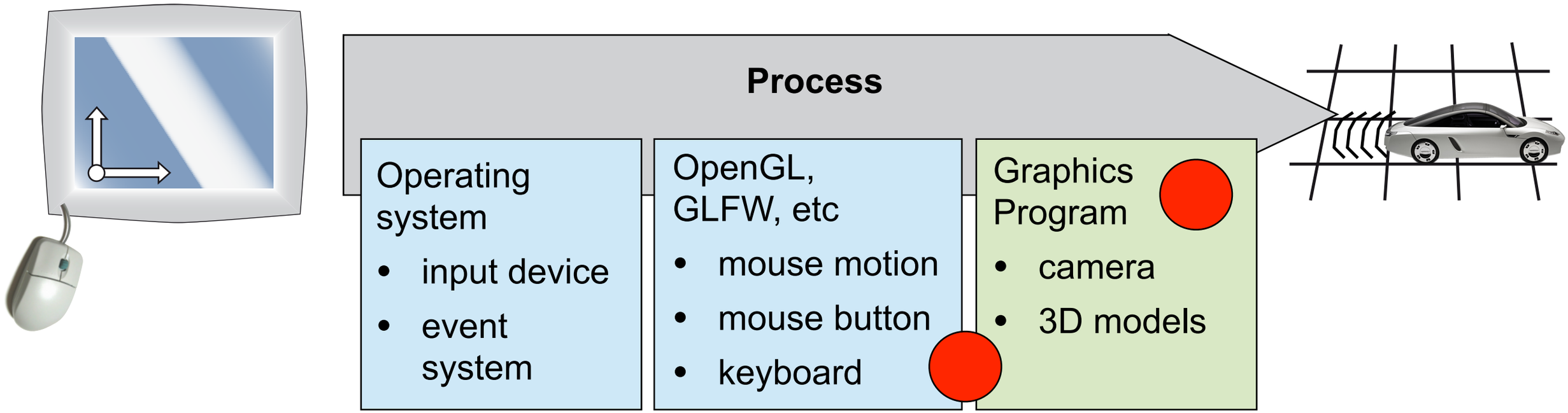
Test if we are within the triangle

Calculate the intersection point

$$\mathbf{p}_i = \mathbf{p}_a + (\mathbf{p}_b - \mathbf{p}_a)t$$

Window System

OpenGL & Interaction



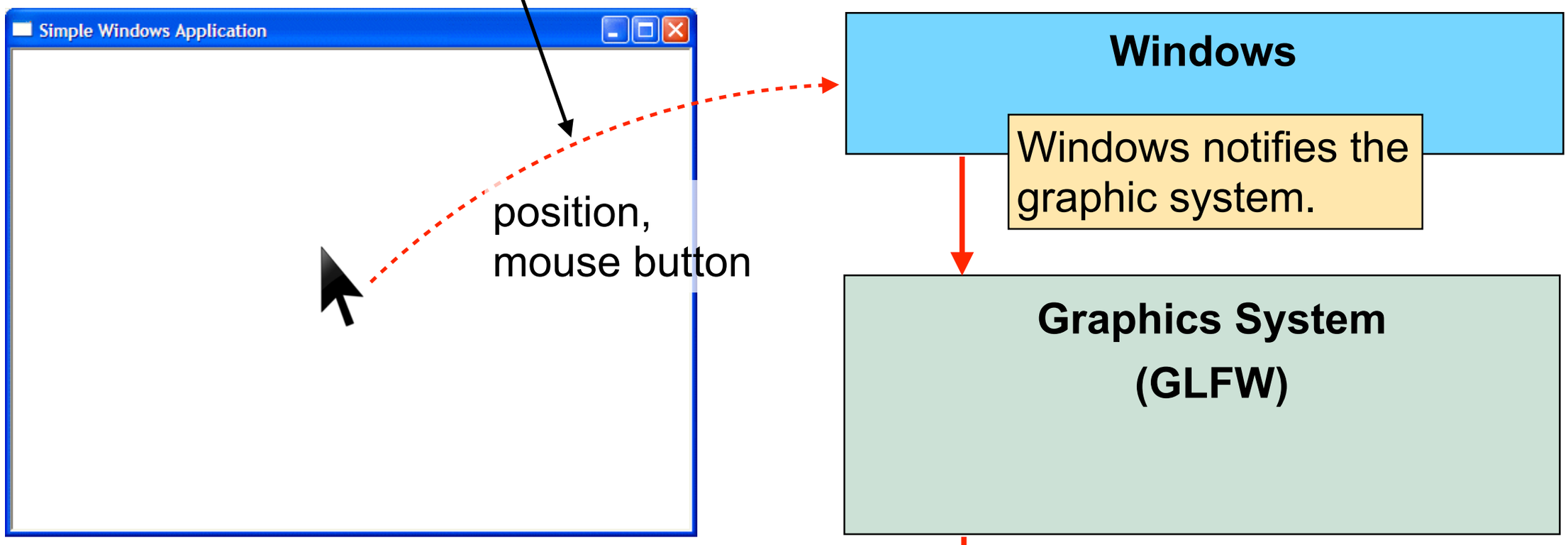
We work on two ends:

First we need to program an event handling system as interface between our application and the underlying framework

Second, we have to program "the Interactor", the mechanism that transfers mouse movements into a camera position / orientation.

Concept

Operating system manages the resource "mouse" and notice the mouse interaction.



The graphics system invokes your function and executes the code.

```
void myFunction( int mouse button, int position)
{
    Interaction: do something, e.g. trackball interaction.
}
```

Interaction Functions

ARLAB

You have to implement and provide a set of functions that execute the interactions.

```
void myMouseMotionFunction( int mouse_id, int position)
{
    Interaction: do something, e.g. trackball interaction.
}
```

```
void myMouseButtonFunction( int mouse_button, int mouse_button)
{
    Interaction: do something, e.g. trackball interaction.
}
```

```
void myKeyboardFunction( int window, int key, int mode, .......)
{
    Interaction: do something else....
}
```

- The function names are on you.
- The signature of the function is given and expected (API documentation).
- You register your functions before you start to render images.

Glfw functions to set callbacks

ARLAB

Search for function on in the Input Handling section of GLFW

http://www.glfw.org/docs/latest/group__input.html

```
GLFWcursorposfun glfwSetCursorPosCallback( GLFWwindow * window,
                                           GLFWcursorposfun cbfun )
```

This function sets the cursor position callback of the specified window, which is called when the cursor is moved

```
GLFWmousebuttonfun glfwSetMouseButtonCallback( GLFWwindow * window,
                                               GLFWmousebuttonfun cbfun )
```

This function sets the mouse button callback of the specified window, which is called when a mouse button is pressed or released.

```
GLFWkeyfun glfwSetKeyCallback( GLFWwindow * window, GLFWkeyfun cbfun)
```

This function sets the key callback of the specified window, which is called when a key is pressed, repeated or released.

Example

ARLAB

```
// This is the callback we'll be registering with GLFW for keyboard handling.
// The only thing we're doing here is setting up the window to close when we press ESC
void keyboard_callback(GLFWwindow* window, int key, int scancode, int action, int mods)
{
    bool move = false;

    if (key == GLFW_KEY_ESCAPE && action == GLFW_PRESS)
    {
        glfwSetWindowShouldClose(window, GL_TRUE);
    }
    else if(key == 87 && action == GLFW_REPEAT) // key w
    {
        //cout <<  "key w pressed" << endl;
        g_tranform_sphere = g_tranform_sphere *
                            glm::translate(glm::vec3(0.0, g_delta, 0.0f));
        sphere->setMatrix(g_tranform_sphere);
        move = true;
    }

[....]
```

Example

ARLAB

```
// This is the callback we'll be registering with GLFW for keyboard handling.
// The only thing we're doing here is setting up the window to close when we press ESC
void keyboard_callback(GLFWwindow* window, int key, int scancode, int action, int mods)
{
    bool move = false;

    if (key == GLFW_KEY_ESCAPE && action == GLFW_PRESS)
    {
        glfwSetWindowShouldClose(window, GL_TRUE);
    }
    else if(key == 87 && action == GLFW_REPEAT) // key w
    {
        //cout <<  "key w pressed" << endl;
        g_tranform_sphere = g_tranform_sphere *
                            glm::translate(glm::vec3(0.0, g_delta, 0.0f));
        sphere->setMatrix(g_tranform_sphere);
        move = true;
    }

[....]
```

GLFW returns the ASCII code of the key the user presses

Hex	Dec	Char		Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char
0x00	0	NULL	null	0x20	32	Space	0x40	64	@	0x60	96	`
0x01	1	SOH	Start of heading	0x21	33	!	0x41	65	A	0x61	97	a
0x02	2	STX	Start of text	0x22	34	"	0x42	66	B	0x62	98	b
0x03	3	ETX	End of text	0x23	35	#	0x43	67	C	0x63	99	c
0x04	4	EOT	End of transmission	0x24	36	$	0x44	68	D	0x64	100	d
0x05	5	ENQ	Enquiry	0x25	37	%	0x45	69	E	0x65	101	e
0x06	6	ACK	Acknowledge	0x26	38	&	0x46	70	F	0x66	102	f
0x07	7	BELL	Bell	0x27	39	'	0x47	71	G	0x67	103	g
0x08	8	BS	Backspace	0x28	40	(	0x48	72	H	0x68	104	h
0x09	9	TAB	Horizontal tab	0x29	41	)	0x49	73	I	0x69	105	i
0x0A	10	LF	New line	0x2A	42	*	0x4A	74	J	0x6A	106	j
0x0B	11	VT	Vertical tab	0x2B	43	+	0x4B	75	K	0x6B	107	k
0x0C	12	FF	Form Feed	0x2C	44	,	0x4C	76	L	0x6C	108	l
0x0D	13	CR	Carriage return	0x2D	45	-	0x4D	77	M	0x6D	109	m
0x0E	14	SO	Shift out	0x2E	46	.	0x4E	78	N	0x6E	110	n
0x0F	15	SI	Shift in	0x2F	47	/	0x4F	79	O	0x6F	111	o
0x10	16	DLE	Data link escape	0x30	48	0	0x50	80	P	0x70	112	p
0x11	17	DC1	Device control 1	0x31	49	1	0x51	81	Q	0x71	113	q
0x12	18	DC2	Device control 2	0x32	50	2	0x52	82	R	0x72	114	r
0x13	19	DC3	Device control 3	0x33	51	3	0x53	83	S	0x73	115	s
0x14	20	DC4	Device control 4	0x34	52	4	0x54	84	T	0x74	116	t
0x15	21	NAK	Negative ack	0x35	53	5	0x55	85	U	0x75	117	u
0x16	22	SYN	Synchronous idle	0x36	54	6	0x56	86	V	0x76	118	v
0x17	23	ETB	End transmission block	0x37	55	7	0x57	87	W	0x77	119	w
0x18	24	CAN	Cancel	0x38	56	8	0x58	88	X	0x78	120	x
0x19	25	EM	End of medium	0x39	57	9	0x59	89	Y	0x79	121	y
0x1A	26	SUB	Substitute	0x3A	58	:	0x5A	90	Z	0x7A	122	z
0x1B	27	FSC	Escape	0x3B	59	;	0x5B	91	[	0x7B	123	{
0x1C	28	FS	File separator	0x3C	60	<	0x5C	92	\	0x7C	124	|
0x1D	29	GS	Group separator	0x3D	61	=	0x5D	93	]	0x7D	125	}
0x1E	30	RS	Record separator	0x3E	62	>	0x5E	94	^	0x7E	126	~
0x1F	31	US	Unit separator	0x3F	63	?	0x5F	95	_	0x7F	127	DEL

ARLAB

ASCII Table

GLFW returns the capital character values by default.

American **S**tandard **C**ode for **I**nformation **I**nterchange

VRAC|HCI

IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

Example

ARLAB

```
// This is the callback we'll be registering with GLFW for keyboard handling.
// The only thing we're doing here is setting up the window to close when we press ESC
void keyboard_callback(GLFWwindow* window, int key, int scancode, int action, int mods)
{
    bool move = false;

    if (key == GLFW_KEY_ESCAPE && action == GLFW_PRESS)
    {
        glfwSetWindowShouldClose(window, GL_TRUE);
    }
    else if(key == 87 && action == GLFW_REPEAT) // key w
    {
        //cout <<  "key w pressed" << endl;
        g_tranform_sphere = g_tranform_sphere *
                            glm::translate(glm::vec3(0.0, g_delta, 0.0f));
        sphere->setMatrix(g_tranform_sphere);
        move = true;
    }

[....]
```

GLFW returns the ASCII code of the key the user presses

Key and button actions

From the GLFW doc webpage

http://www.glfw.org/docs/latest/group__input.html

Key and button actions

```
#define GLFW_RELEASE 0
```
The key or mouse button was released. More...

```
#define GLFW_PRESS 1
```
The key or mouse button was pressed. More...

```
#define GLFW_REPEAT 2
```
The key was held down until it repeated. More...

These are information that you typically find in the API documentation.

Thank you!

Questions

Rafael Radkowski, Ph.D.
Iowa State University
Virtual Reality Applications Center
1620 Howe Hall
Ames, Iowa 50011, USA

+1 515.294.5580
+1 515.294.5530(fax)

rafael@iastate.edu
http://arlab.me.iastate.edu

www.linkedin.com/in/rradkowski

ARLAB

VRAC|HCI

IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY