ARLAB

ME/CprE/ComS 557

Computer Graphics and Geometric Modeling

Picking / Scissor Test

December 1st, 2015

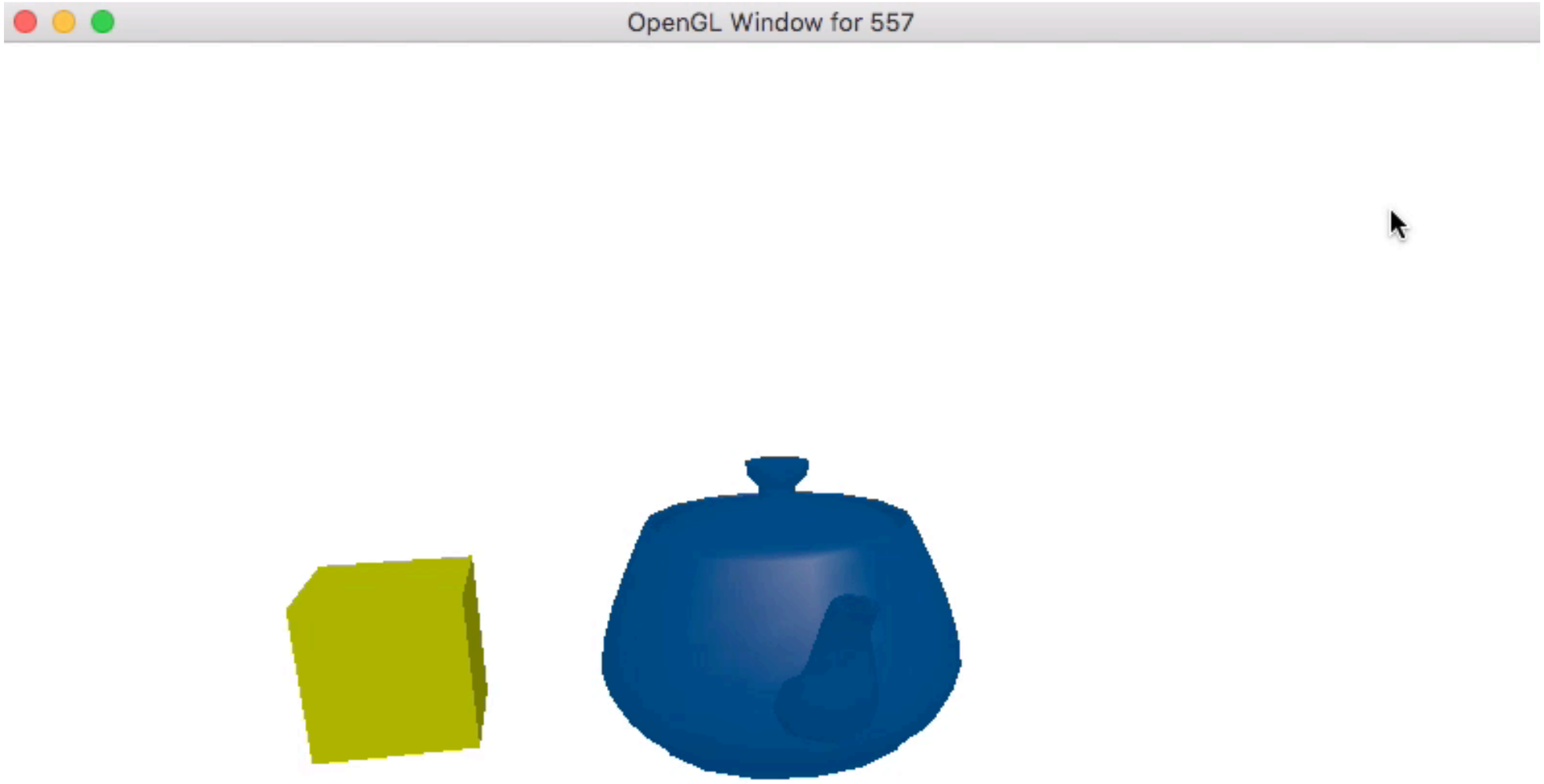
Rafael Radkowski

IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

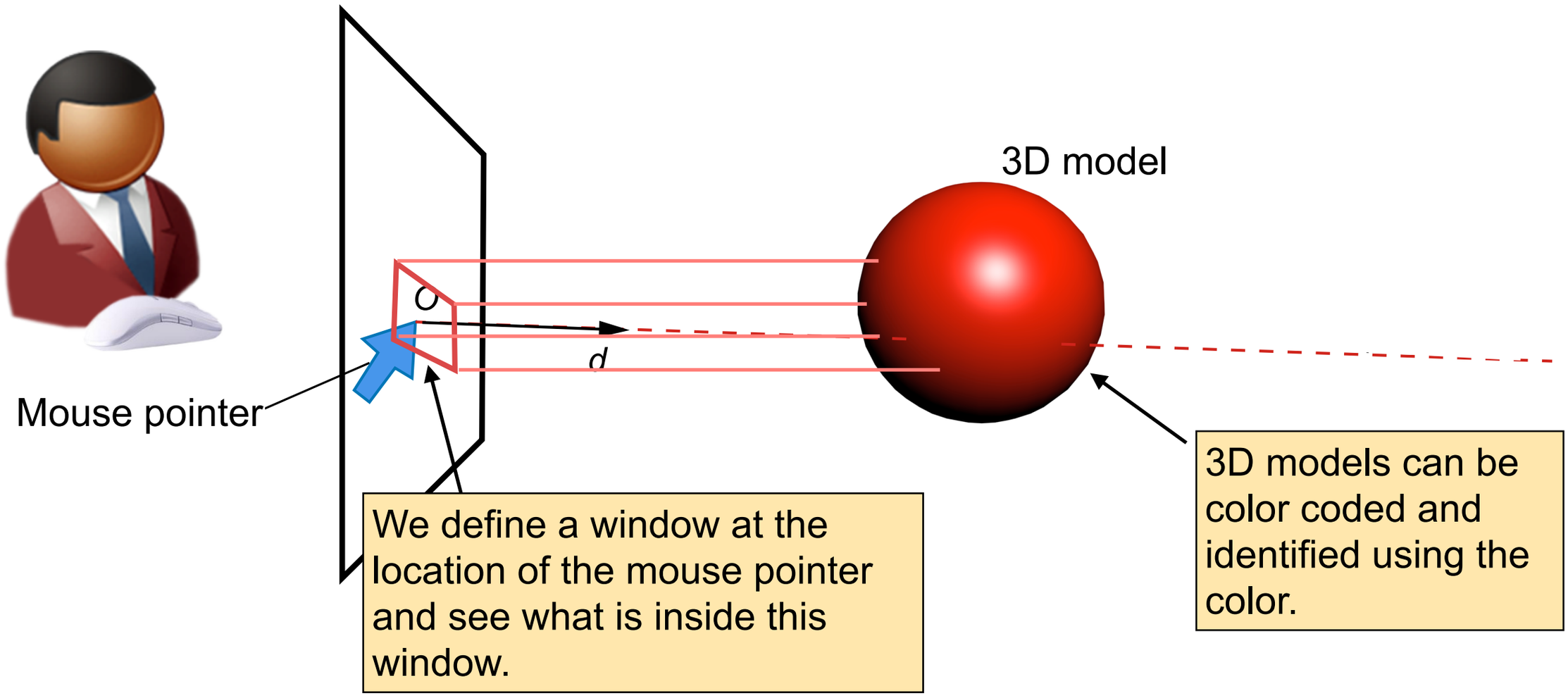
Content

- Introduction
- Selection vs. Render Mode
- Object identification
- Scissor Test
- Scissor Test and Picking
- Read values
- Code Example

Introduction



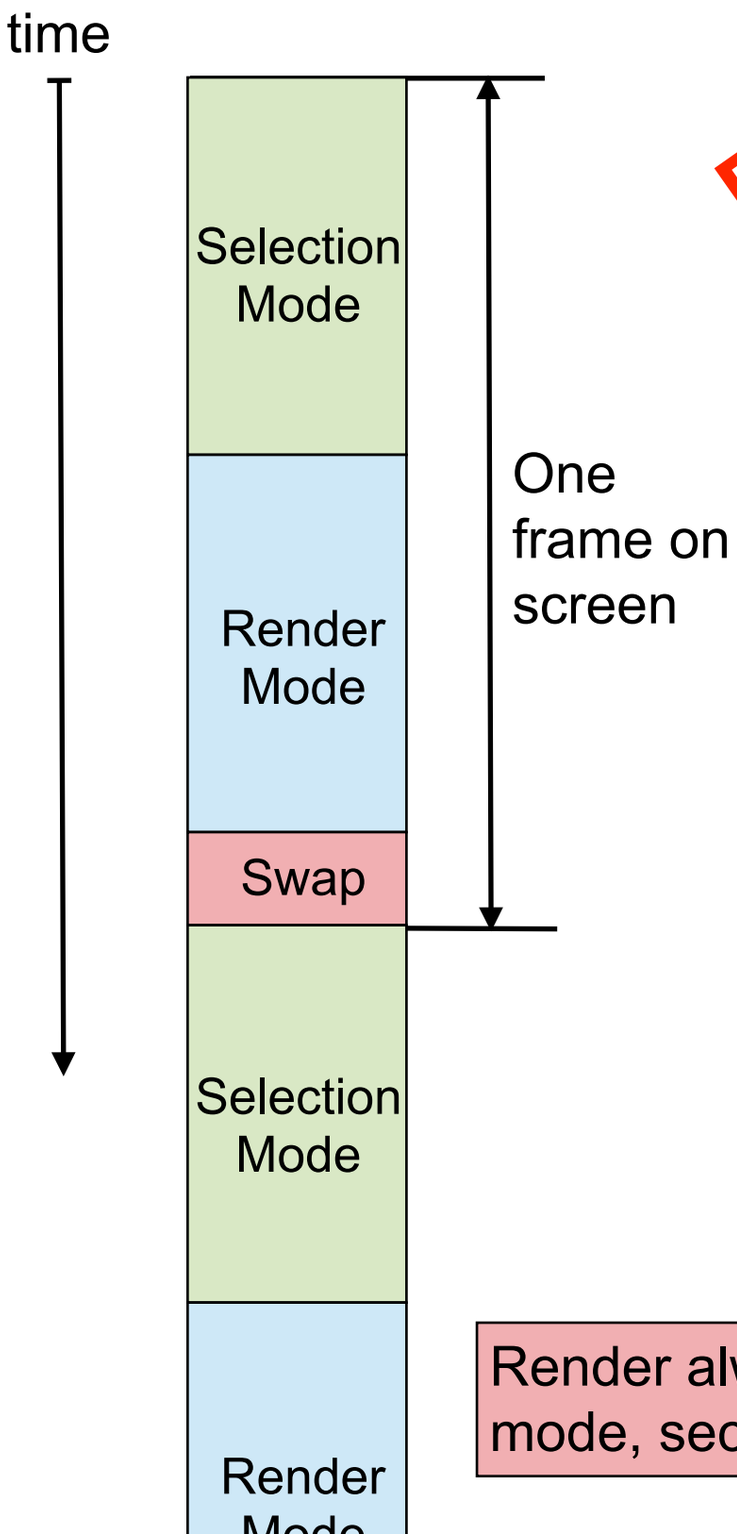
Picking with a Window



We need

- read the mouse pointer position
- define a window at the mouse pointer position and render only into this window.
- Read the color and associate the object with a color.

Selection vs. Render Mode



Deprecated!

OpenGL distinguish a **Render Mode** and a **Selection Mode.**

- Render Mode: the rendered scene is displayed on screen.
- Selection Mode: no frames are rendered into frame buffer, instead, the names of the objects are fetched.

To switch to render mode:

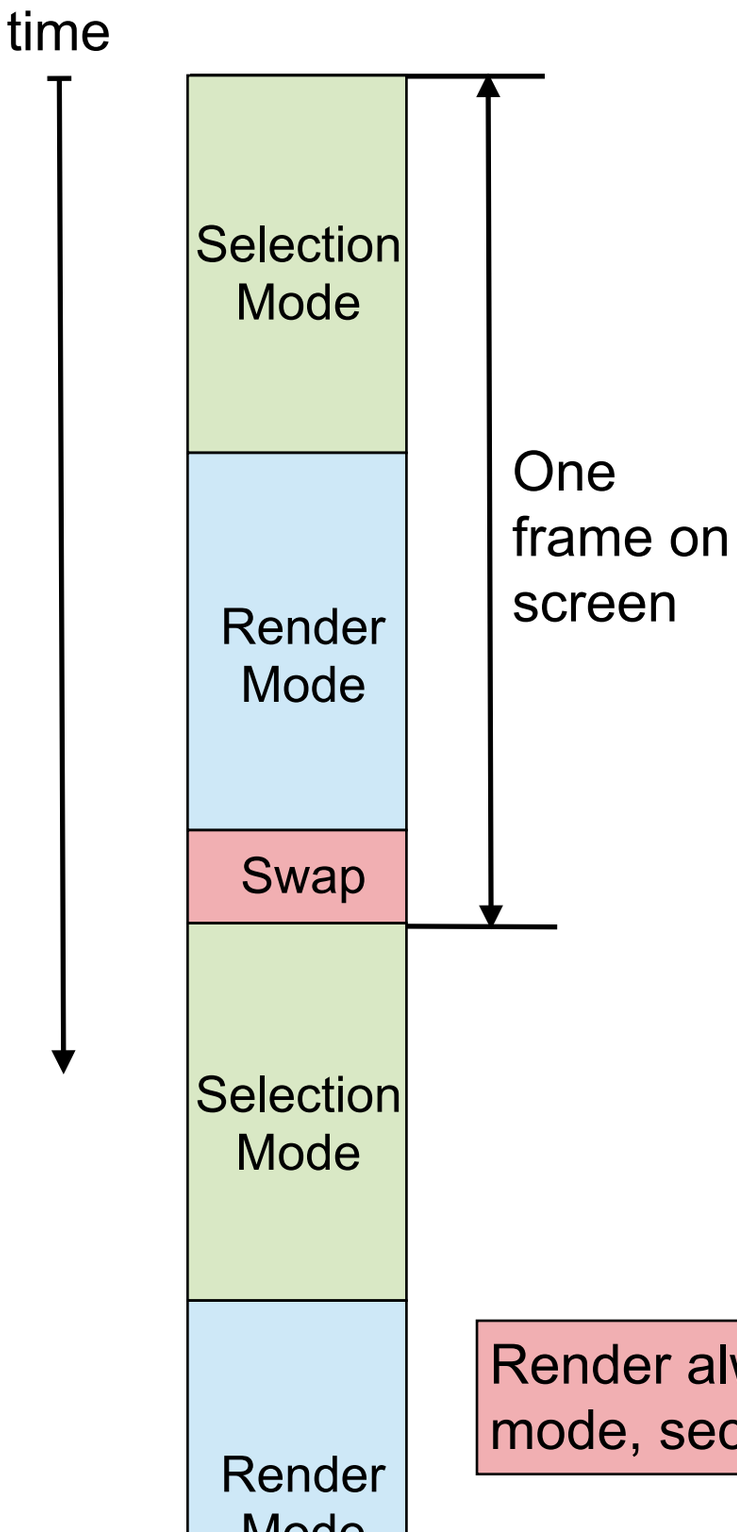
```
GLint glRenderMode(GLenum mode)
```

Parameters:

- mode: the render mode
  - GL_RENDER: render into frame buffer
  - GL_SELECT: render for picking
- return: the number of hits when switching from selection mode to render mode

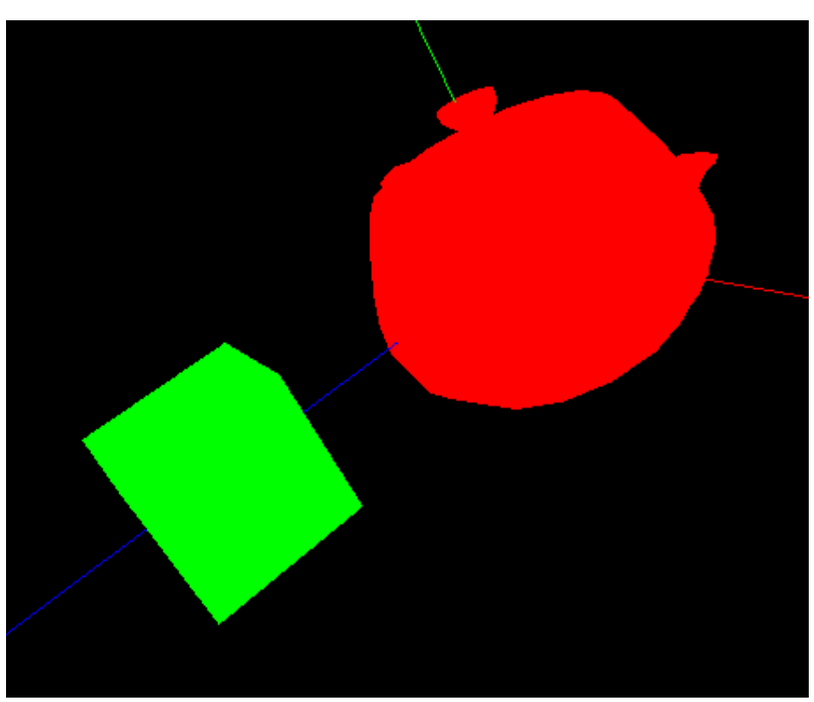
Render always first in selection mode, second in render mode!!

Selection vs. Render Mode

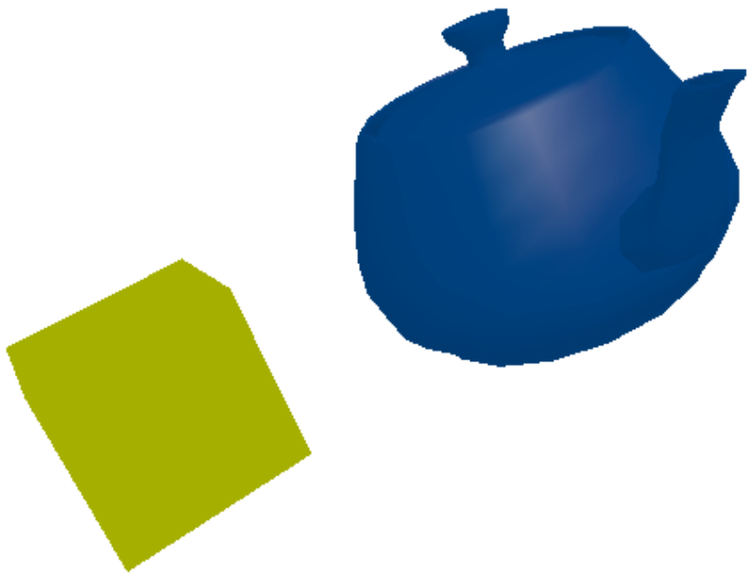


OpenGL does not provide an explicit render mode and selection mode with GLSL (it is not necessary anymore), but we need to implement something similar.

1. Render all objects for picking only. Use the color as id.

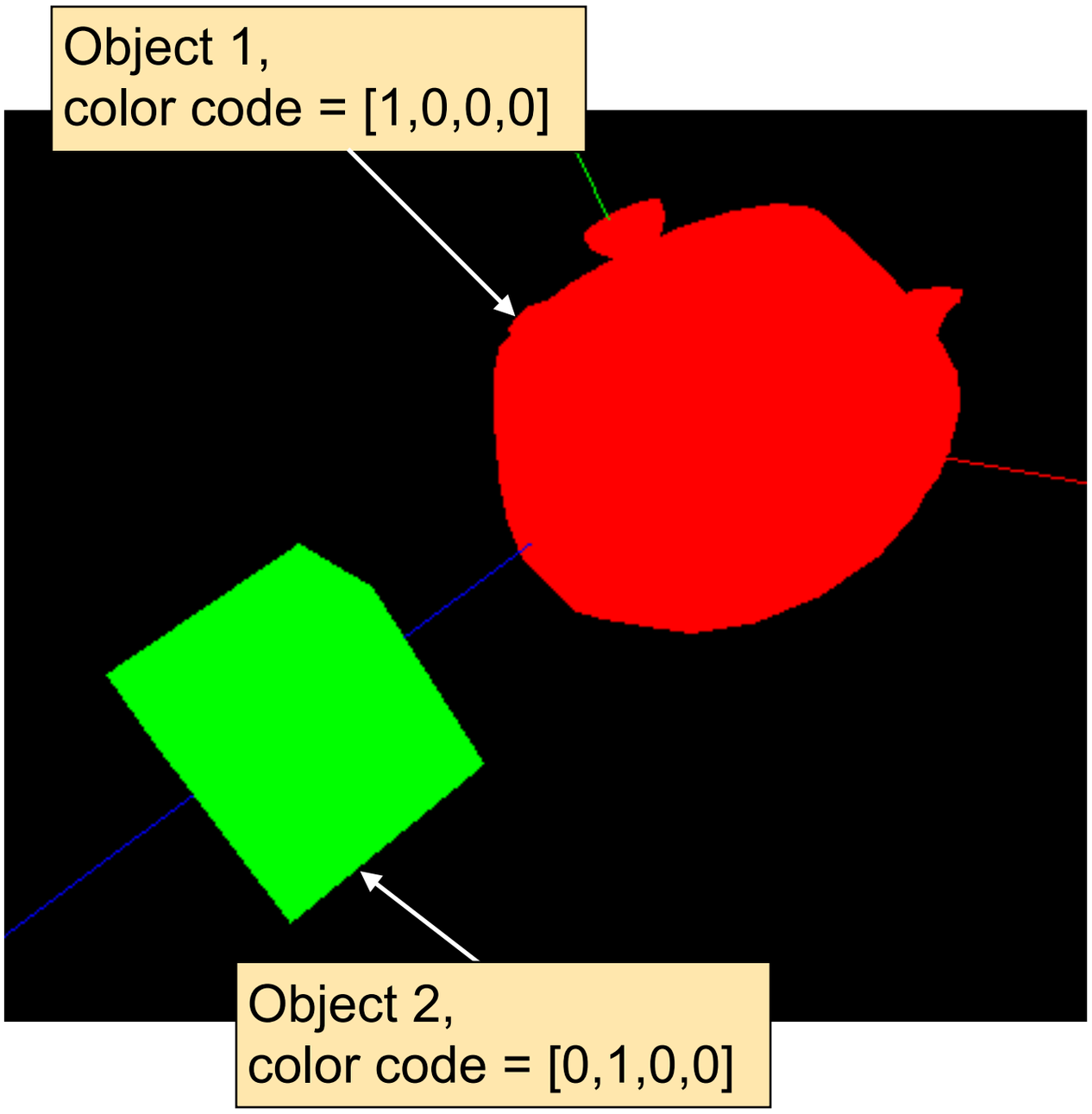


2. Render the regular, visual scene.



Render always first in selection mode, second in render mode!!

Object identification



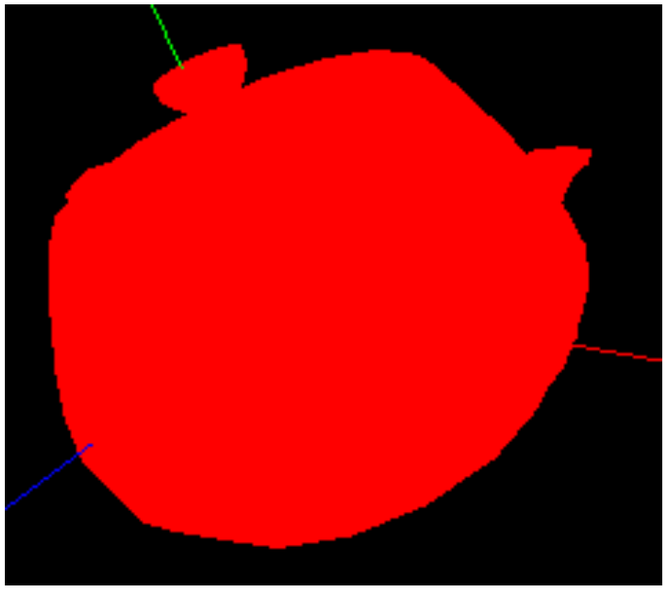
"Visual" identification: We render the object underneath the mouse pointer tip into a tiny window and identify the color.

Therefore, we need to

- encode every object with a unique color
- compute an id from this color
- associate the object with the id and store this id.

ID	Object
8	teapot
4	box
	...

Object identification



Color is encoded in RGB

Red	Green	Blue	Alpha
1	0	0	0

Find a unique color for each object.

One way to obtain an object id is to interpret the color values as bits of an integer.

8 bit integer:

$$N = a_7\ 2^7 + a_6\ 2^6 + a_5\ 2^5 + a_4\ 2^4 + a_3\ 2^3 + a_2\ 2^2 + a_1\ 2^1 + a_0\ 2^0 = \sum_{i=0}^{b} = a_i\ 2^i$$

	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0
bits	0	0	0	0	1	0	0	0
					Red	Green	Blue	Alpha

VRAC|HCI

IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

Object identification

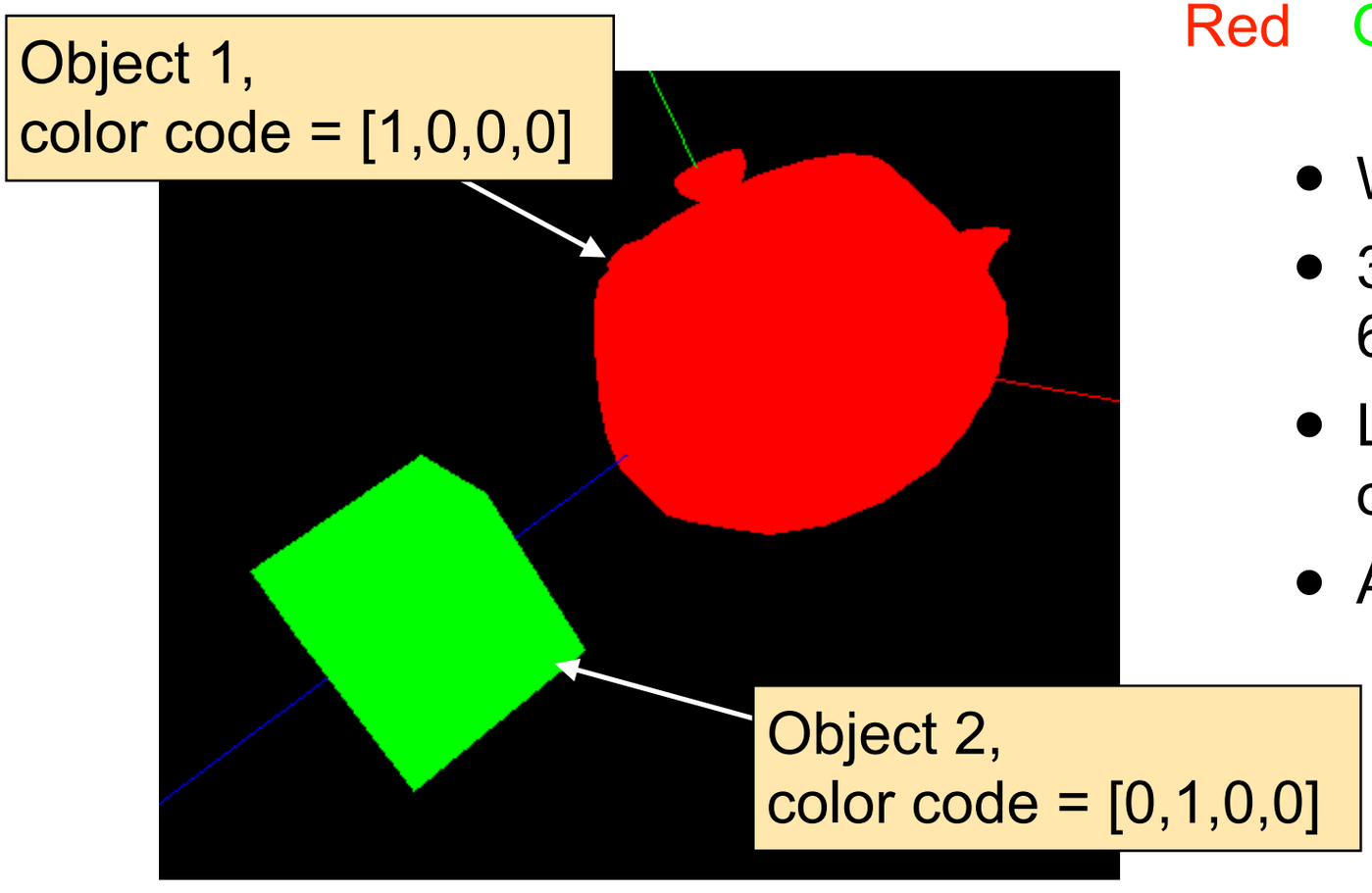
ARLAB

8 bit integer:

$$N = a_7\ 2^7 + a_6\ 2^6 + a_5\ 2^5 + a_4\ 2^4 + a_3\ 2^3 + a_2\ 2^2 + a_1\ 2^1 + a_0\ 2^0 = \sum_{i=0}^{b} = a_i\ 2^i$$

	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	
teapot	0	0	0	0	1	0	0	0	= 8
box	0	0	0	0	0	1	0	0	= 4

Red Green Blue Alpha



- We will not be able to use alpha.
- 3-bits left, thus, we can represent 6 objects + 1 idle state.
- Limited to a small number of objects.
- Advantage, the id is obvious

Object identification

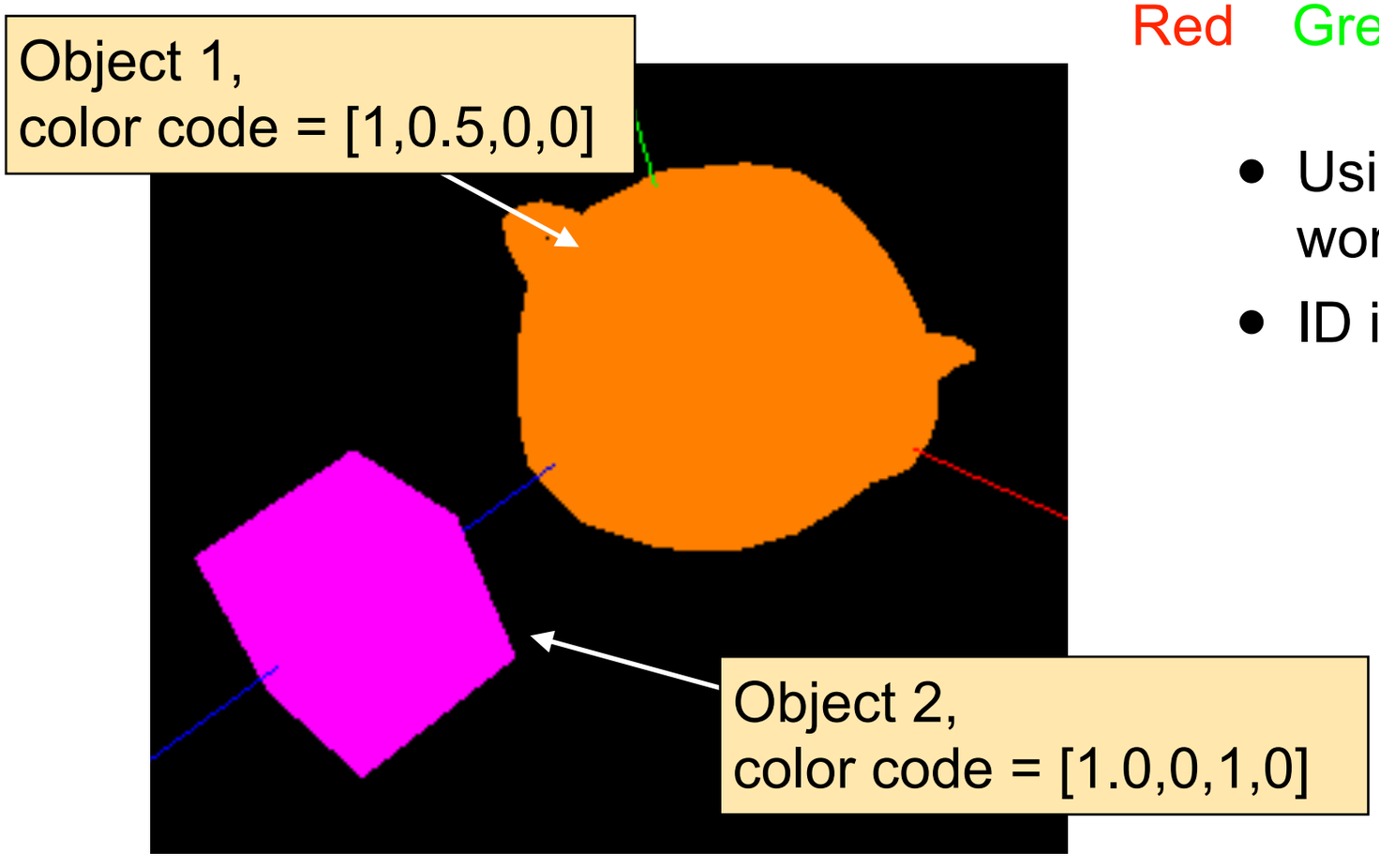
ARLAB

Obtain float values:

$$N = a_7\ 2^7 + a_6\ 2^6 + a_5\ 2^5 + a_4\ 2^4 + a_3\ 2^3 + a_2\ 2^2 + a_1\ 2^1 + a_0\ 2^0 = \sum_{i=0}^{b} = a_i\ 2^i$$

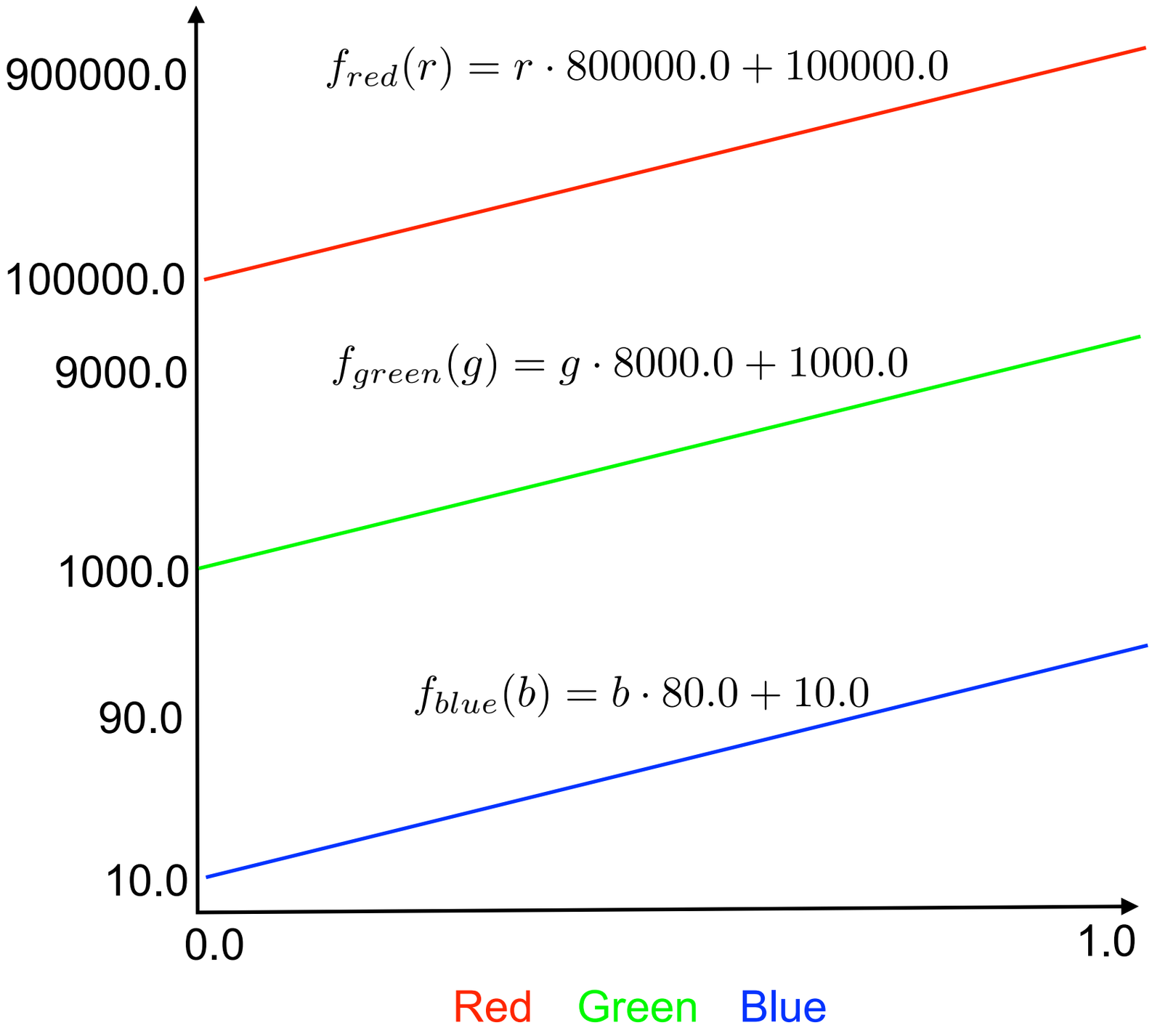
	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	
teapot	0	0	0	0	1.0	0.5	0	0	= 10
box	0	0	0	0	1.0	0.0	1.0	0	= 10

Red Green Blue Alpha



- Using the same equation does not work.
- ID is not unique anymore

Linear functions



$$ID = f_{red} + f_{green} + f_{blue}$$

Example:

RGB = {0.8, 0.4, 0.2}

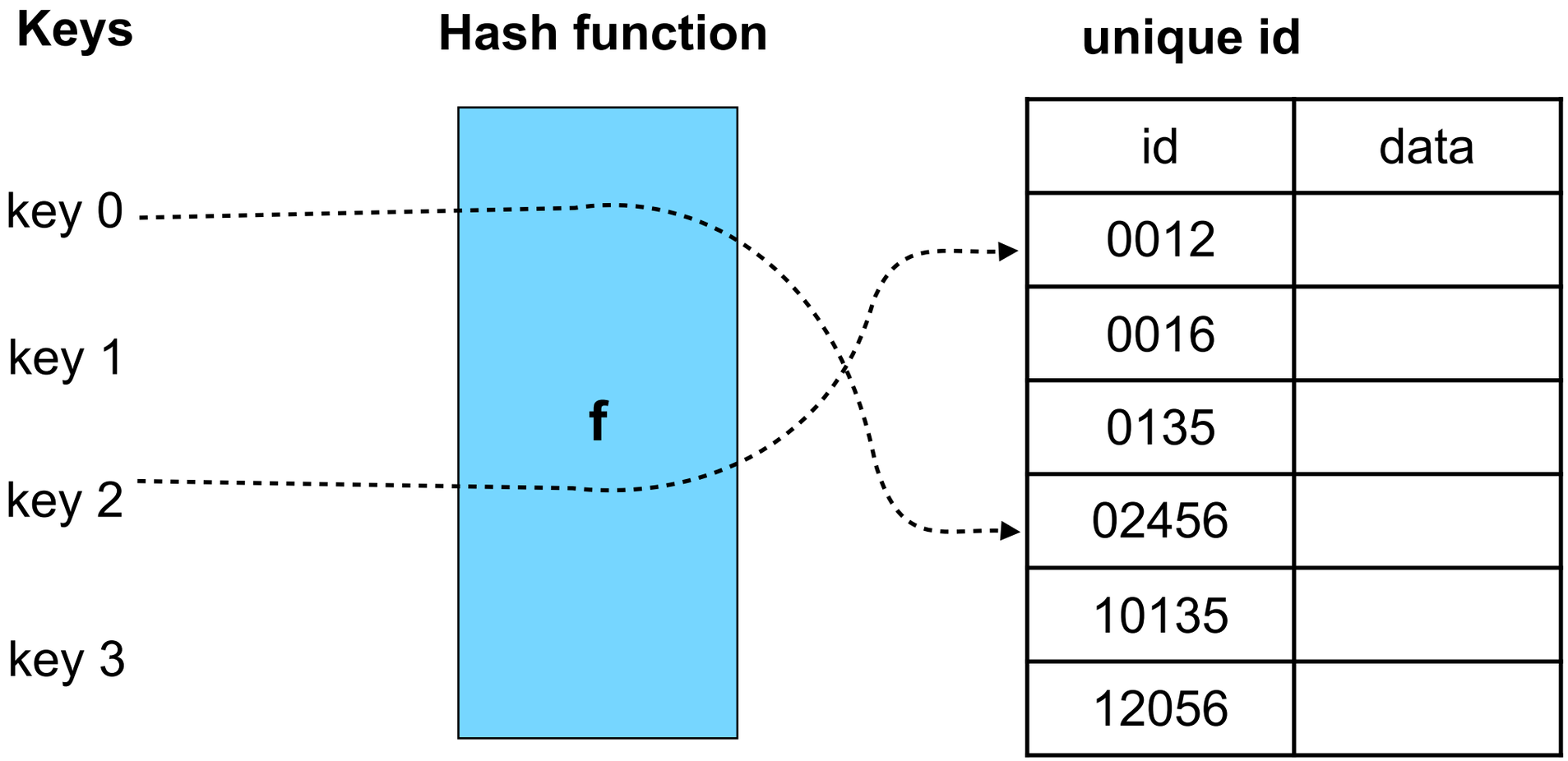
ID = 744226

- The color value only alters two digits.
- BUT: with 10^5 as max id, the color values can only change in 0.1 increments.

Hash Key / Hash Function

This is just an outlook.

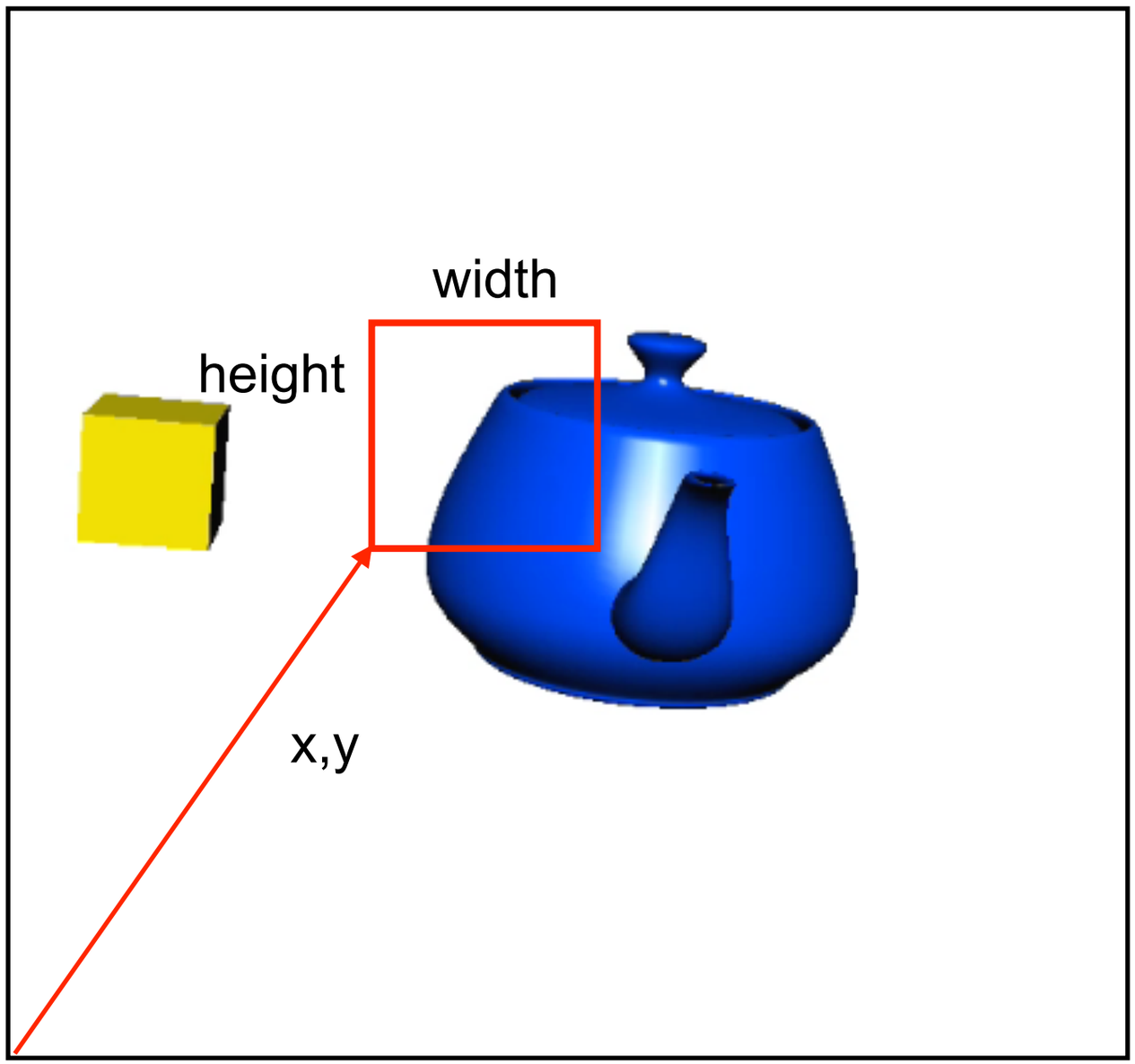
A hash function is a special function to generate a **unique key** from a given set of values.
It is used for data structure, database and cryptographic applications.



Collisions remain a challenge!!

Scissor Test

The scissor test is a per-object processing operation that discards fragments that fall outside of a certain rectangular portion of the screen.



glScissor

Define a scissor window

```
void glScissor (GLint x , GLint y , GLsizei width , GLsizei height );
```

Parameters:

x, y: the start position of the window in pixel.

width, height: the size of the window in pixel

The scissor test is a graphics card function, thus, it needs to be enabled / disabled when required, using

- glEnable(GL_SCISSOR_TEST)
- glDisable(GL_SCISSOR_TEST)

The argument GL_SCISSOR_TEST is associated with the scissor test.

- You can enable / disable the test for each object.
- The scissor window size and position can also be changed per object.

Program Structure

ARLAB

Header

```
int main(int argc, const char * argv[])
{
        [....]

        [....]
```

Program init

```
    while
```

main loop

- **Clear the window**
- glEnable(GL_SCISSOR_TEST)
- **Render objects**
- glDisable(GL_SCISSOR_TEST)
- **Render objects**
- **Swap buffers**

```
}
```

VRAC|HCI IOWA STATE UNIVERSITY OF SCIENCE AND TECHNOLOGY

Program Structure

ARLAB

Header

```
int main(int argc, const char * argv[])
{
        [....]
        glScissor (... )
        [....]
```

Program init

You can change the scissor window whenever you need to change it.

```
while
```

main loop

Clear the window

glScissor (...)

glEnable(GL_SCISSOR_TEST)

Render objects

glDisable(GL_SCISSOR_TEST)

Render objects

Swap buffers

```
}
```

State Machine

OpenGL uses a state machine as computational model.

The function

```
glScissor (GLint x , GLint y , GLsizei width , GLsizei height )
```

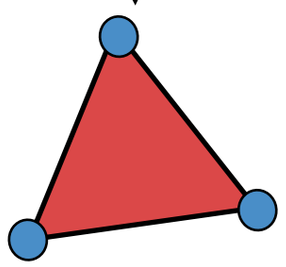
changes the parameters
used for the scissor test.

glEnable(GL_SCISSOR_TEST)

glDisable(GL_SCISSOR_TEST)

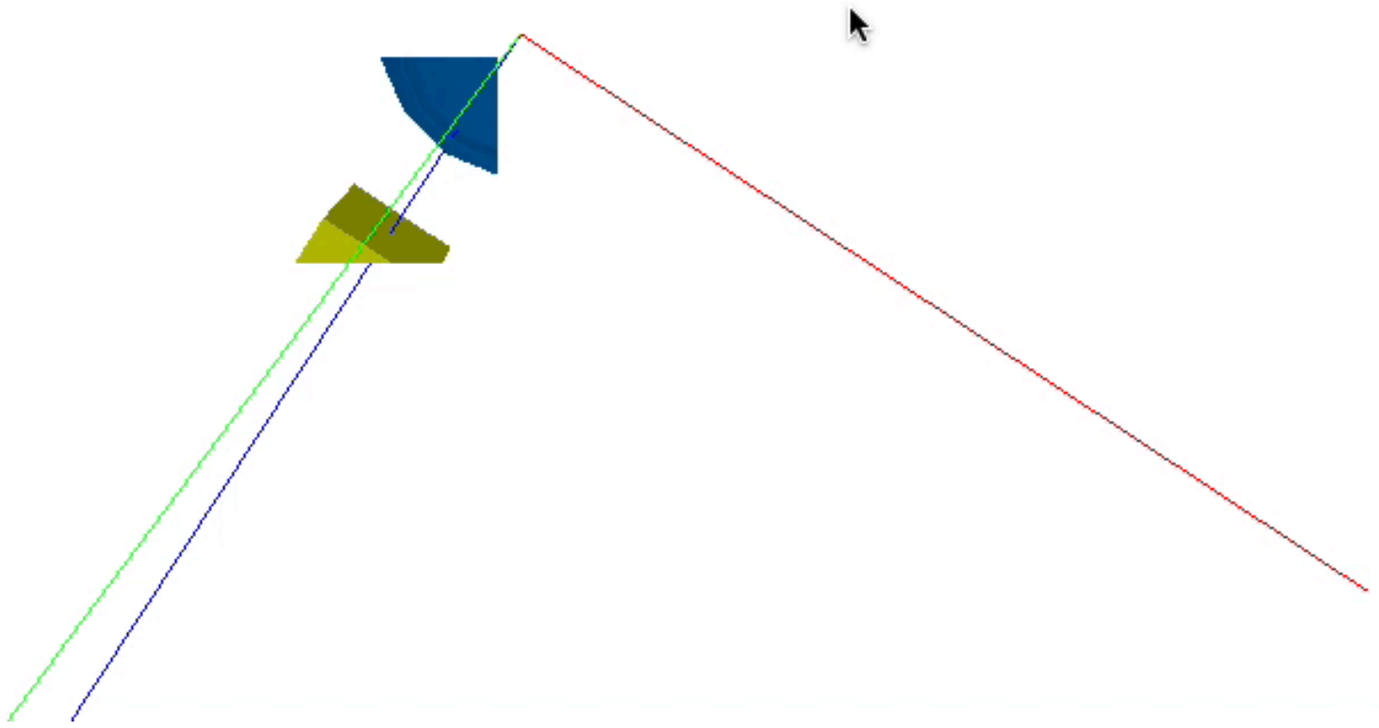
enables and disables the capability.

Name	Value
width	10
height	10
x	300
y	200

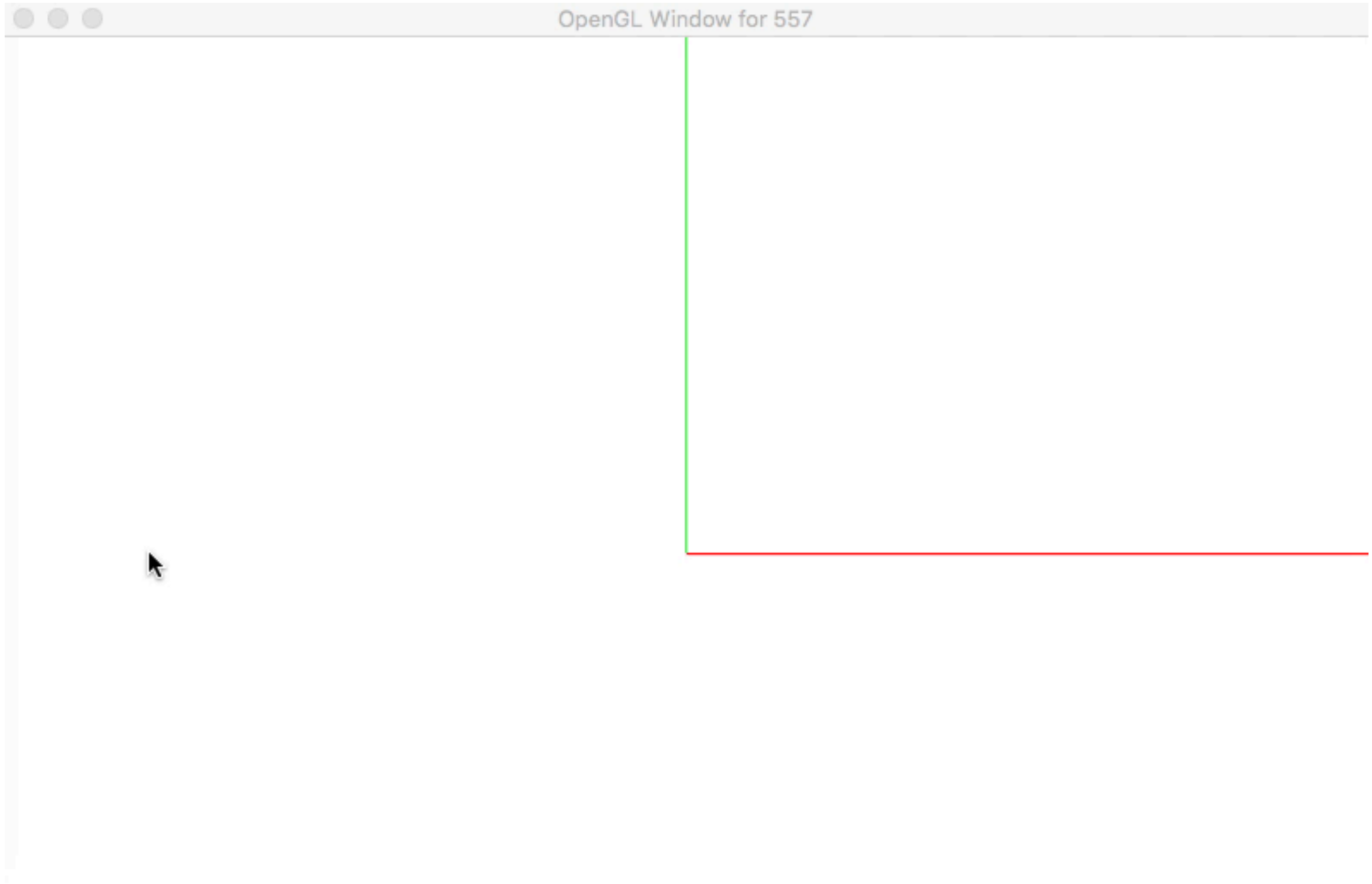


enabled state
disabled state

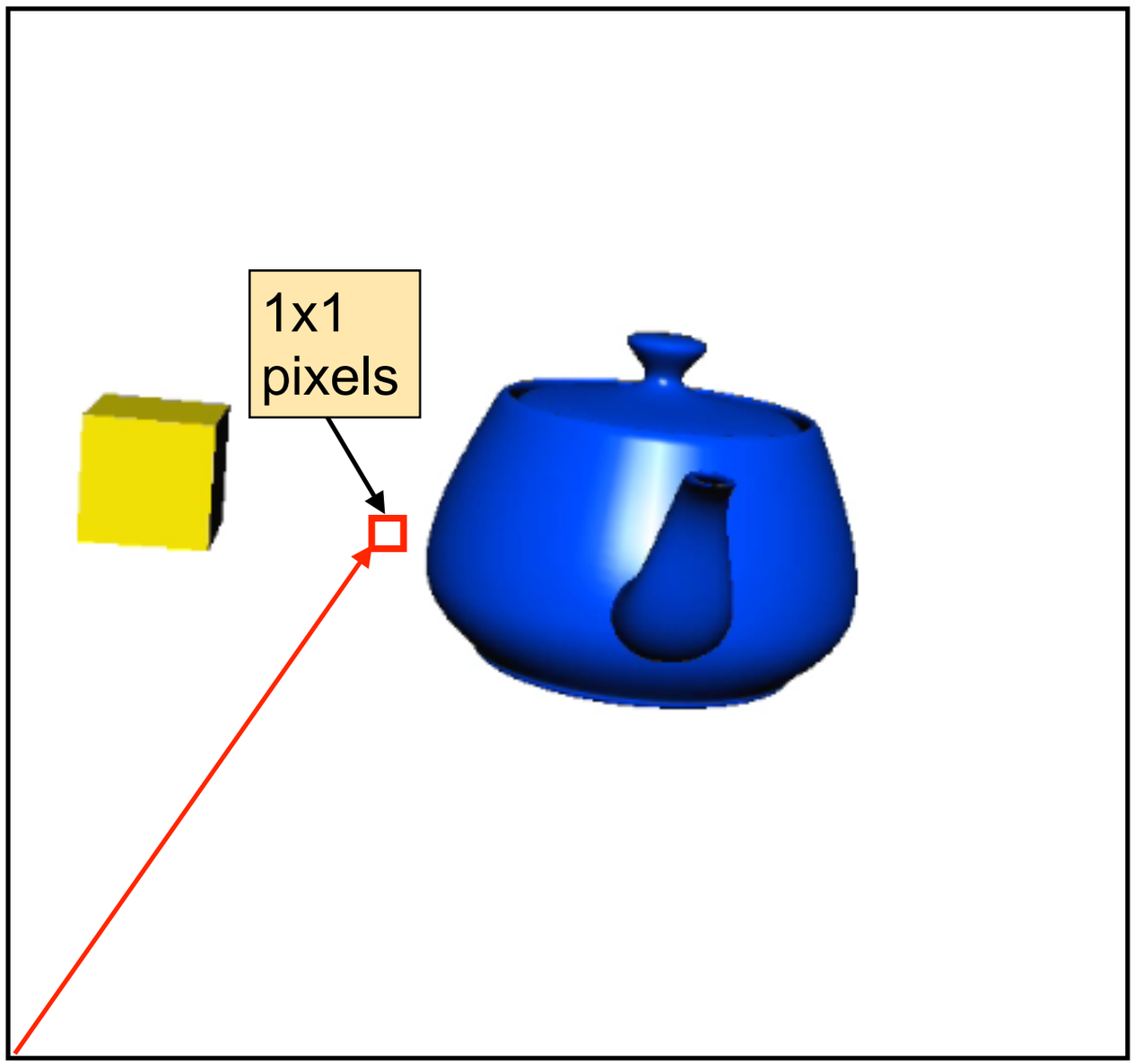
Video



Video

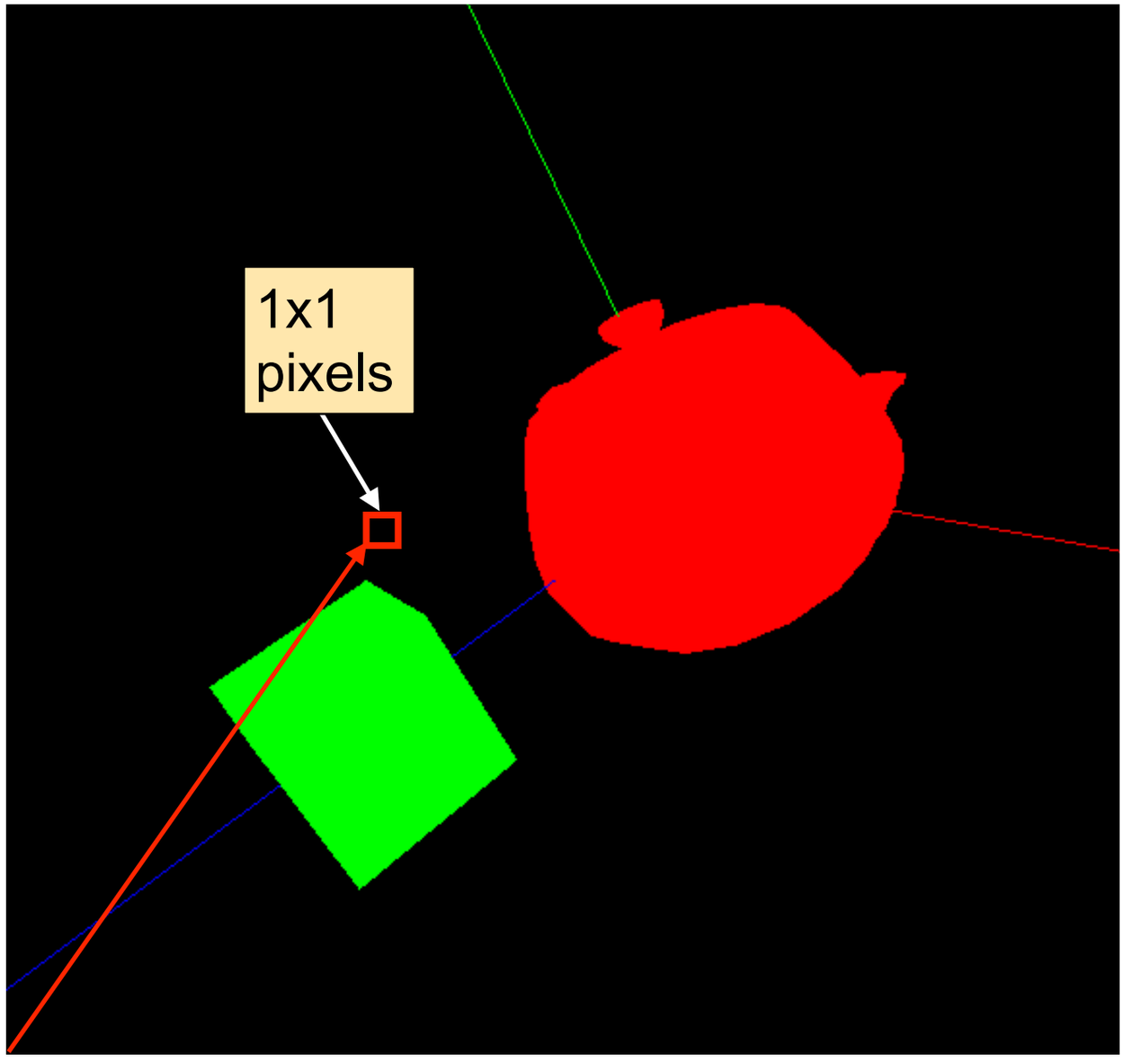


Scissor Test and Picking



To pick an object, we reduce the size of the window to 1x1 pixels and move it along with the mouse pointer.

Scissor Test and Picking



To pick an object, we reduce the size of the window to 1x1 pixels and move it along with the mouse pointer.

We obtain the color information while rendering in "Select"-mode.

glReadPixels

ARLAB

```
void glReadPixels( GLint x, GLint y, GLsizei width,
GLsizei height, GLenum format, GLenum type,
GLvoid * data);
```

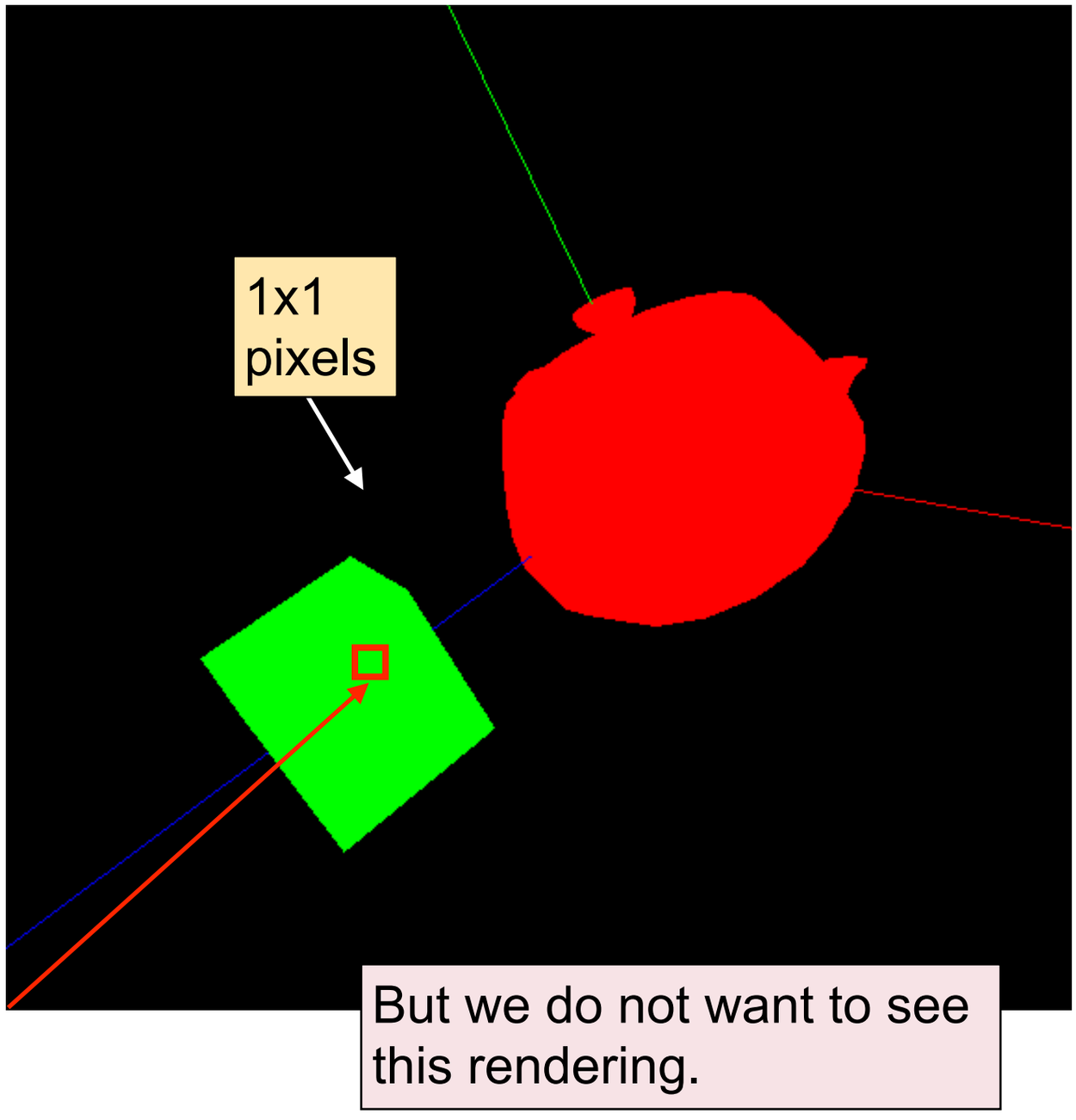
Parameters:

- x, y - Specify the window coordinates of the first pixel that is read from the frame buffer.
- width, height - Specify the dimensions of the pixel rectangle.
- format - Specifies the format of the pixel data. The following symbolic values are accepted: GL_ALPHA, GL_RGB, and GL_RGBA.
- type - Specifies the data type of the pixel data. Must be one of GL_UNSIGNED_BYTE, GL_UNSIGNED_SHORT_5_6_5, GL_UNSIGNED_SHORT_4_4_4_4, or GL_UNSIGNED_SHORT_5_5_5_1.
- data - Returns the pixel data.

Example:

```
float col[4];
glReadPixels(100, 100, 1, 1, GL_RGBA, GL_FLOAT, &col);
```

Scissor Test and Picking



To pick an object, we reduce the size of the window to 1x1 pixels and move it along with the mouse pointer.

We obtain the color information while rendering in "Select"-mode.

```
float col[4];
col = {0.0, 0.0, 1.0, 0.0}
```

Note, the alpha value will always be 0.0, even if you define a different one.

Program Structure

ARLAB

Header

```
int main(int argc, const char * argv[])
{
```

main loop

while

Clear the window

Read mouse position into x, y

glEnable(GL_SCISSOR_TEST)

glScissor (x, y, 1, 1)

Render objects in select mode

glDisable(GL_SCISSOR_TEST)

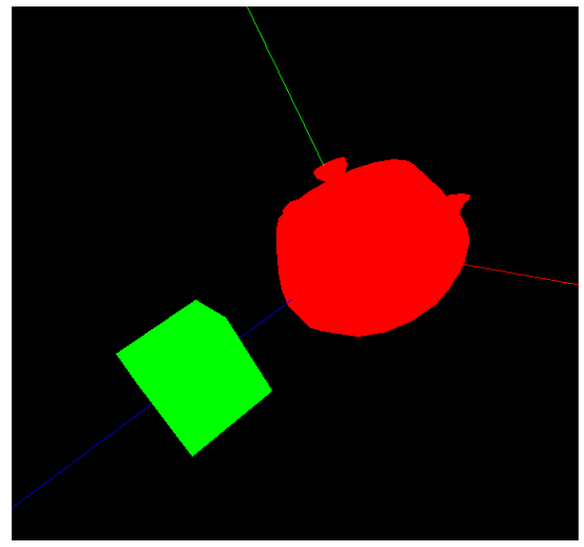
glReadPixels( )

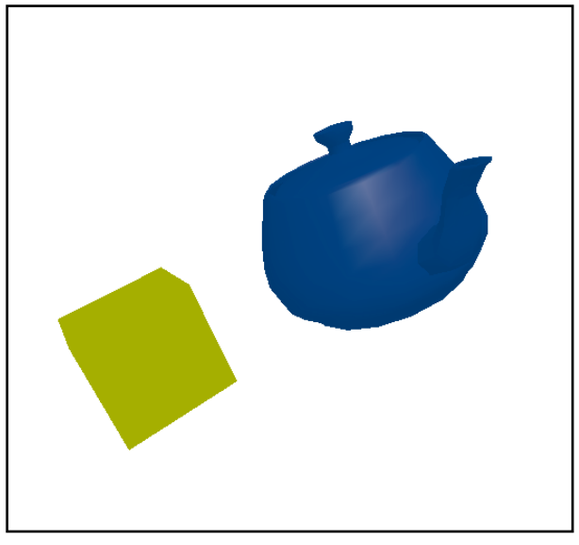
Clear the window

Render objects

Swap buffers

```
}
```





Program Structure

We render the objects in select mode, however, we clear the buffer again after we obtain the color information. Thus, the objects in select mode never show up on screen.

main loop

```
while
```

Clear the window

Read mouse position into x, y

```
glEnable(GL_SCISSOR_TEST)
glScissor (x, y, 1, 1 )
```

Render objects in select mode

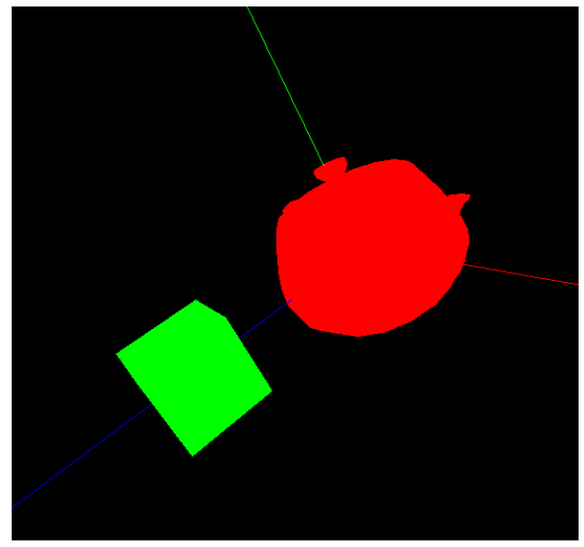
```
glDisable(GL_SCISSOR_TEST)
glReadPixels(  )
```

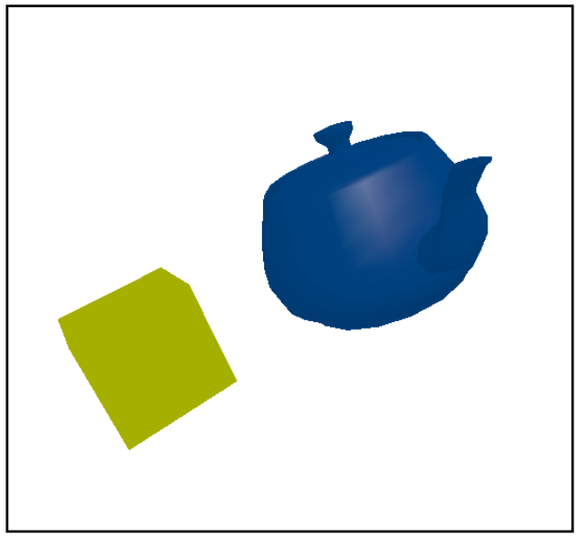
Clear the window

Render objects in render mode

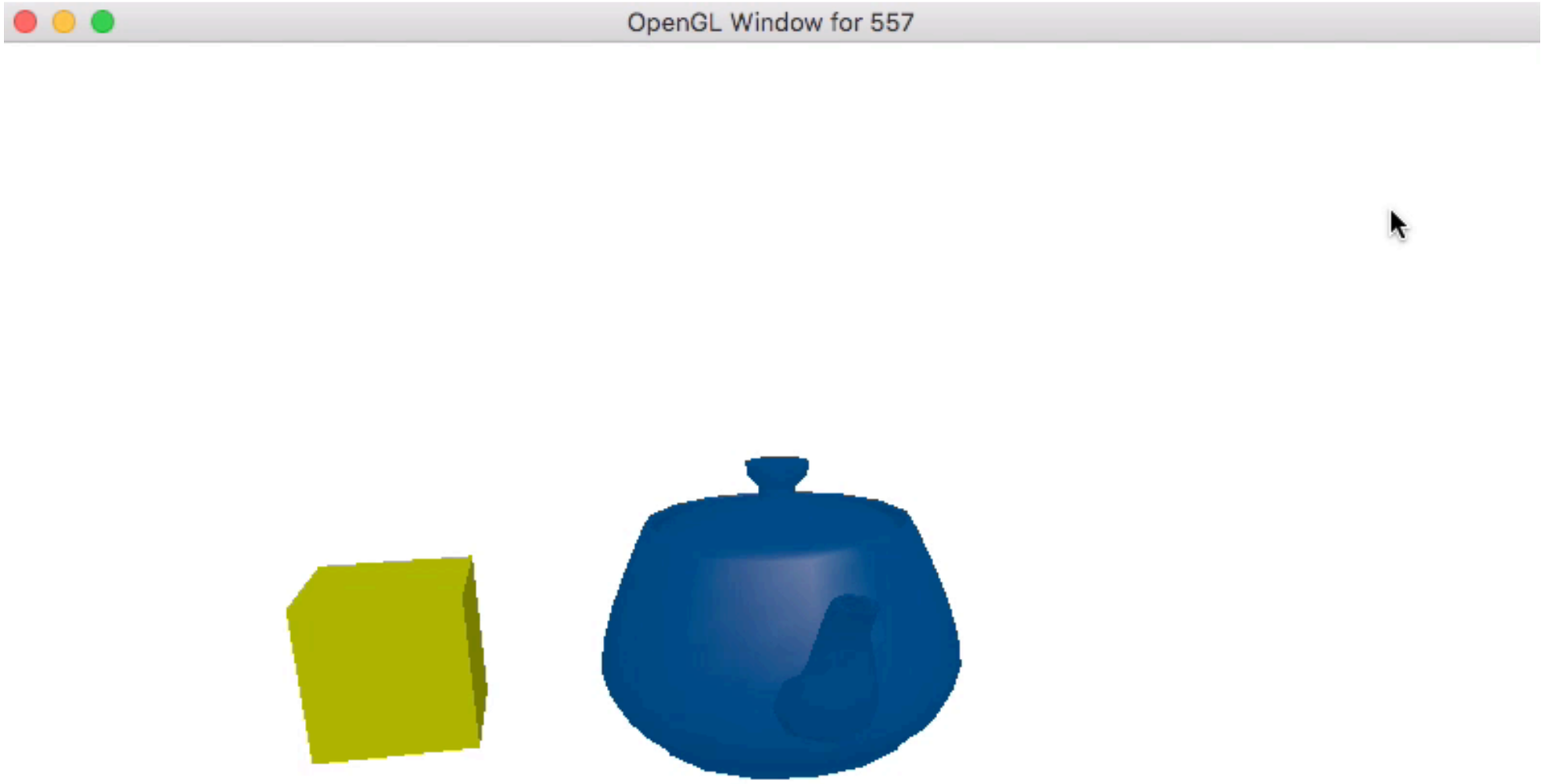
Swap buffers

```
}
```





Video



Code Example

Example Code

main program: **main_picking.cpp**

The entire scissor test is part of the main file.

Shader code: **multi_vertex_lights_ext.vs/.fs**

Functionality to change the color and to switch between select and render mode is implemented as shader program.

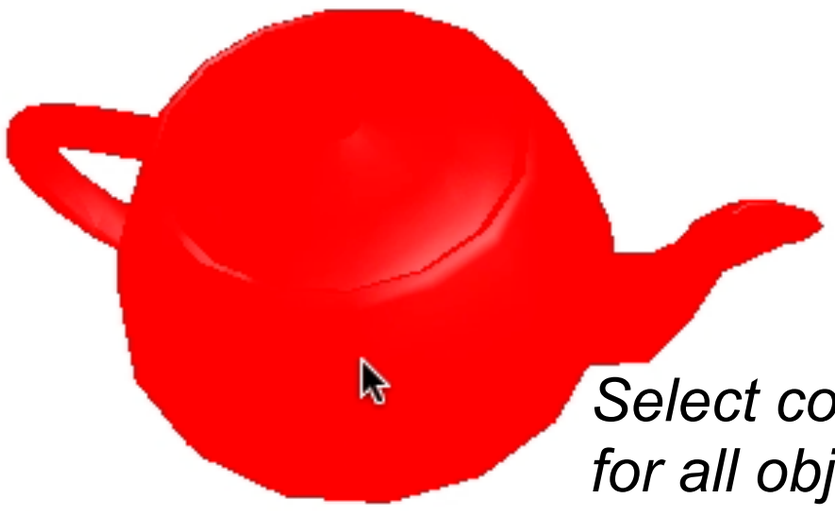
multi_vertex_lights_ext.vs

The code implements a per-vertex light renderer.

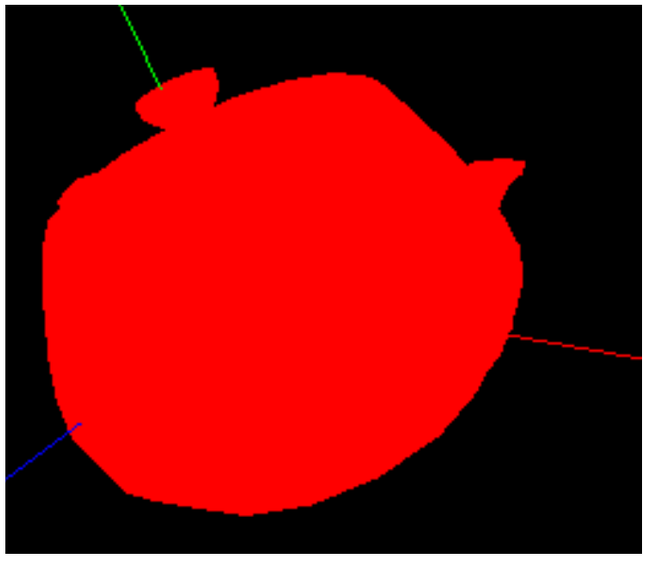
Four new variables have been introduced for picking.

```
// variable to distinguish between renderer for
// the selection buffer and the regular renderer
uniform bool select_mode;
uniform vec4 select_color_id;
uniform bool is_selected; // to indicate that this object has been selected
```

```
// the color that shows up when an object has been picked
const vec3 select_color = vec3(1.0,0.0,0.0);
```



Select color is red for all objects



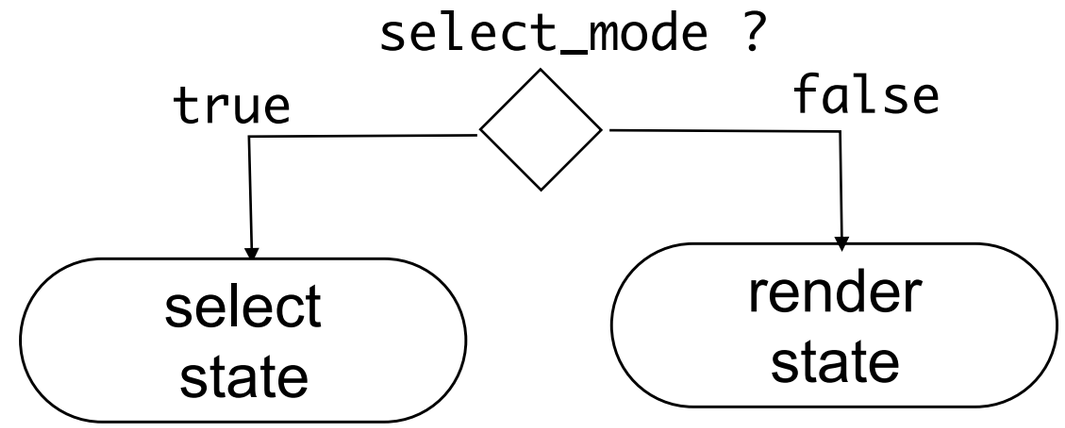
select_color_id, the example teapot is red = 8

multi_vertex_lights_ext.vs

```
if(select_mode==false)
{
    // Calculate the reflection

   [....]


}
else
{
    // Pass the color
    pass_Color =  vec4(select_color_id);
}
```



multi_vertex_lights_ext.vs

We use the function useLight to calculate the reflection, thus, the color for one light source.

```
vec4 useLight( ..... )
{

    // Here: all the light calculations happen.

    // Change the color of the object when it is selected.
    if(is_selected == true)
    {
        out_diffuse_color = select_color;
        out_ambient_color = select_color;
    }

}
```

Note, is_selected is a uniform variable. Thus, we change the value in our host program.

main_picking.cpp

ARLAB

```
// This variable allows us to keep track of the selected object.
// Start with your idle state.
// 0 = no object
int g_selected_object_id = 0;

GLObjectObj* loadedModel1 = NULL; // this is a teapot
GLObjectObj* loadedModel2 = NULL; // this is a box
```

main_picking.cpp

ARLAB

```
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////
//// Prepare the shader for the scissor test
//// 1. Activate the shader programs
//// 2. Set a select color and remember the position of the select-switch.
//// 3. Set the values.

glUseProgram(apperance_0->getProgram());
int l0 = glGetUniformLocation(apperance_0->getProgram(), "select_mode");
int sel0 = glGetUniformLocation(apperance_0->getProgram(), "is_selected");
glUniform1i(l0, false);
glUniform1i(sel0, false);
glUniform4f(glGetUniformLocation(apperance_0->getProgram(), "select_color_id"),
            1.0,0.0,0.0,1.0 );

glUseProgram(apperance_1->getProgram());
int l1 = glGetUniformLocation(apperance_1->getProgram(), "select_mode");
int sel1 = glGetUniformLocation(apperance_1->getProgram(), "is_selected");
glUniform1i(l1, false);
glUniform1i(sel1, false);
glUniform4f(glGetUniformLocation(apperance_1->getProgram(), "select_color_id"),
            0.0,1.0,0.0,1.0 );
glUseProgram(0);
```

main_picking.cpp

ARLAB

```
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////
//// Prepare the shader for the scissor test
//// 1. Activate the shader programs
//// 2. Set a select color and remember the position of the select switch
//// 3. Set the values.

glUseProgram(apperance_0->getProgram());
int l0 = glGetUniformLocation(apperance_0->getProgram(), "select_mode");
int sel0 = glGetUniformLocation(apperance_0->getProgram(), "is_selected");
glUniform1i(l0, false);
glUniform1i(sel0, false);
glUniform4f(glGetUniformLocation(apperance_0->getProgram(), "select_color_id"),
            1.0,0.0,0.0,1.0 );

glUseProgram(apperance_1->getProgram());
int l1 = glGetUniformLocation(apperance_1->getProgram(), "select_mode");
int sel1 = glGetUniformLocation(apperance_1->getProgram(), "is_selected");
glUniform1i(l1, false);
glUniform1i(sel1, false);
glUniform4f(glGetUniformLocation(apperance_1->getProgram(), "select_color_id"),
            0.0,1.0,0.0,1.0 );
glUseProgram(0);
```

enable the glsl program

fetch the location of the additional variables

set the values

Do this for each object with a different id / color

```
////////////////////////////////////////////////////////////
// For selection.
// FIRST, RENDER IN SELECT MODE
// 1. start the scissor test
glEnable(GL_SCISSOR_TEST);

// 2. Set the window with window size 1x1
// 600 is the size of the frame, make sure you know it.
glScissor(GetMouseX(), 600-GetMouseY(), 1, 1);

// 3. Render the first object
// Switch to seletion mode and render the first object
glUseProgram(apperance_0->getProgram());
glUniform1i(l0, true);

// render
loadedModel1->draw();
glUniform1i(l0, false); // and switch to regular mode.

// 4. Render the second object
// Switch to seletion mode and render the first object
glUseProgram(apperance_1->getProgram());
glUniform1i(l1, true);

// render
loadedModel2->draw();

// switch back.
glUniform1i(l1, false);

glUseProgram(0);
```

600 is the window height. Mouse coordinates and window coordinates are inverted!!!

Switch back to render-mode. We still need to render the object, thus, we must return to the render mode.

```
// render
loadedModel2->draw();

// switch back.
glUniform1i(l1, false);

glUseProgram(0);

// AT THIS LOCATION, WE HAVE TO RENDER ANY ADDITIONAL OBJECT IN "SELECT"-MODE
// NOTE, GLSL, OPENGL 4.5 DOES NOT PROVIDE A SELECT MODE ANYMORE, WE SIMULATE THIS
MODE.

// 5. Disable the scissor test.
glDisable(GL_SCISSOR_TEST);
```

We must right the color information back as long as it is inside the frame buffer, thus, here.

```
////////////////////////////////////////////////////////////
// Read the color information at that pixel.
float col[4];
glReadPixels(GetMouseX(), 600-GetMouseY(), 1, 1, GL_RGB, GL_FLOAT,&col);
//cout << "COLOR:\t" << col[0] << "\t" << col[1] << "\t" << col[2]  << "\t" <<
col[3] << endl;

int object_id = colorToInteger(col[0], col[1], col[2], col[3]);
//cout << "Found object with id: " << object_id << endl;

////////////////////////////////////////////////////////////
// Process the color information
// Change the color of the selected object
handle_pick(object_id);
```

colorToInteger

ARLAB

```
/**
This functions converts four integer digits, to an integer value.
The digits should be either 0 or 1.
*/
int colorToInteger(int r, int g, int b, int a)
{
    int selected_object_id = 0;

    selected_object_id = selected_object_id << 1;
    selected_object_id |= r;
    selected_object_id = selected_object_id << 1;
    selected_object_id |= g;
    selected_object_id = selected_object_id << 1;
    selected_object_id |= b;
    selected_object_id = selected_object_id << 1;
    selected_object_id |= a;

    return selected_object_id;
}
```

Operator	Symbol	Form	Operation
left shift	<<	x << y	all bits in x shifted left y bits
right shift	>>	x >> y	all bits in x shifted right y bits
bitwise NOT	~	~x	all bits in x flipped
bitwise AND	&	x & y	each bit in x AND each bit in y
bitwise OR	|	x | y	each bit in x OR each bit in y
bitwise XOR	^	x ^ y	each bit in x XOR each bit in y

colorToInteger

ARLAB

$$N = a_7\ 2^7 + a_6\ 2^6 + a_5\ 2^5 + a_4\ 2^4 + a_3\ 2^3 + a_2\ 2^2 + a_1\ 2^1 + a_0\ 2^0 = \sum_{i=0}^{b} = a_i\ 2^i$$

128	64	32	16	8	4	2	1
a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	0
0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	1
0	0	0	0	1	0	1	0

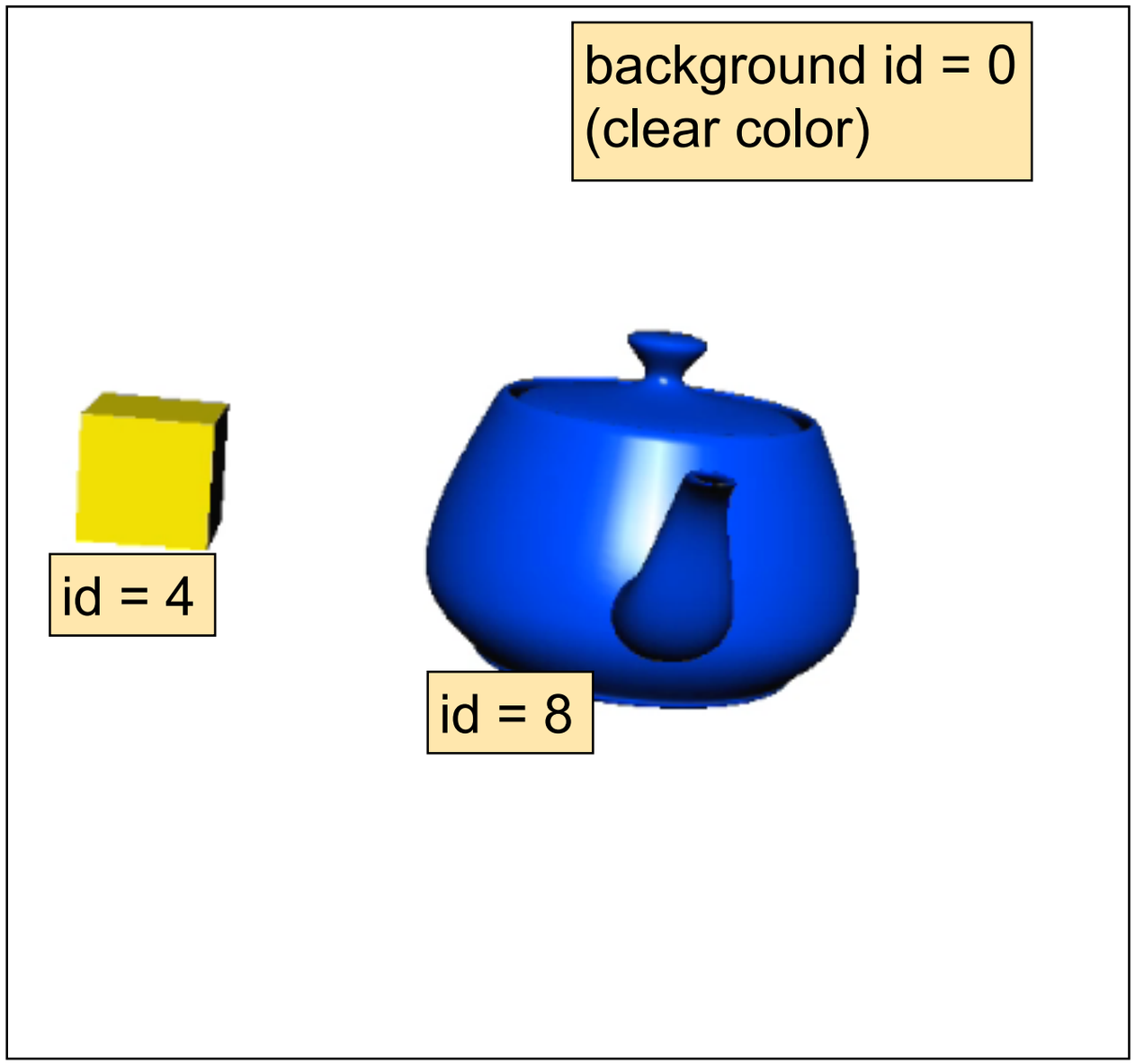
= 10

```
r = 1, g = 0, b = 1

int selected_object_id = 0;
selected_object_id |= r;
selected_object_id << 1;
selected_object_id |= g;
selected_object_id << 1;
selected_object_id |= b;
selected_object_id << 1;
```

VRAC|HCI IOWA STATE UNIVERSITY OF SCIENCE AND TECHNOLOGY

handle_pick



We need to handle three situations:

1. No changes (idle)
2. Change from 0 to id > 0

   Change the color of the selected object
3. Change from id > 0 to id > 0

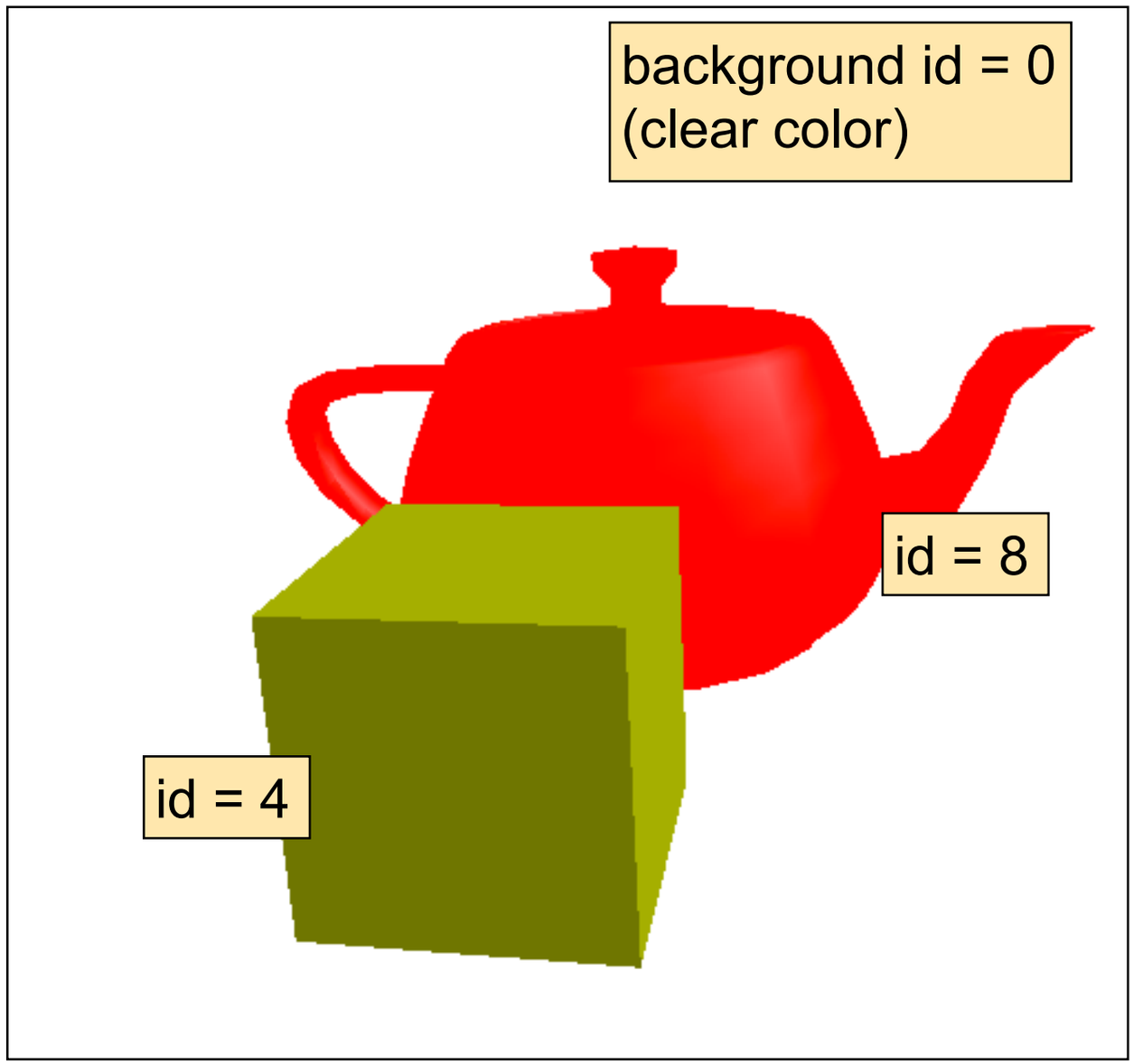
   One object is already selected (color = red).

   Switch the color back to the regular material.

   Change the color of the new selection to red

These situations are not only relevant when changing the color.

handle_pick



We need to handle three situations:

1. No changes (idle)
2. Change from 0 to id > 0

   Change the color of the selected object
3. Change from id > 0 to id > 0

   One object is already selected (color = red).

   Switch the color back to the regular material.

   Change the color of the new selection to red

These situations are not only relevant when changing the color.

handle_pick


```
void handle_pick(int selected_object_id)
{

    // first, detect whether a pick event occured.
    // If the same object is picked of the clear color shows up, we return.
    // No state change.
    if(selected_object_id == g_selected_object_id)
    {
        return;
    }
```

We compare the current id with the last selection to detect changes

```
    // consider, an object is already picked, disable the pick color.
    // g_selected_object_id == 0, means, no object selected
    if(g_selected_object_id > 0)
    {
        [...]
    }
```

If we have detected an id-change, and the current id is larger than 0, we change the color of this object back to its previous material

```
    // Now we change the color of the selected object
    switch(selected_object_id)
    {
        [...]
    }
}
```

We change the color of the current selected object.

handle_pick

```
// consider, an object is already picked, disable the pick color.
// g_selected_object_id == 0, means, no object selected
if(g_selected_object_id > 0)
{
    // Note, this can be optimized/
    // We need a list that keeps an association between model and its id.
    // this can be an additional list or part of the object itself.
    // In this case, and for training reasons (not hiding the code,
    // this is a switch-case control flow.
    switch(g_selected_object_id)
    {
    case 4:
        setSelectColor(loadedModel2->getProgram(), false);
        g_selected_object_id = 0;
        break;
    case 8:
        setSelectColor(loadedModel1->getProgram(), false);
        g_selected_object_id = 0;
        break;
    }
  }
}
```

False deactivates the select-color

handle_pick

```
// Now we change the color of the selected object
switch(selected_object_id)
    switch(g_selected_object_id)
    {
    case 4:
         setSelectColor(loadedModel2->getProgram(), true);
         g_selected_object_id = 0;
         break;
    case 8:
         setSelectColor(loadedModel1->getProgram(), true);
         g_selected_object_id = 0;
         break;
        }
    }
}
```

True activates the select-color

setSelectColor

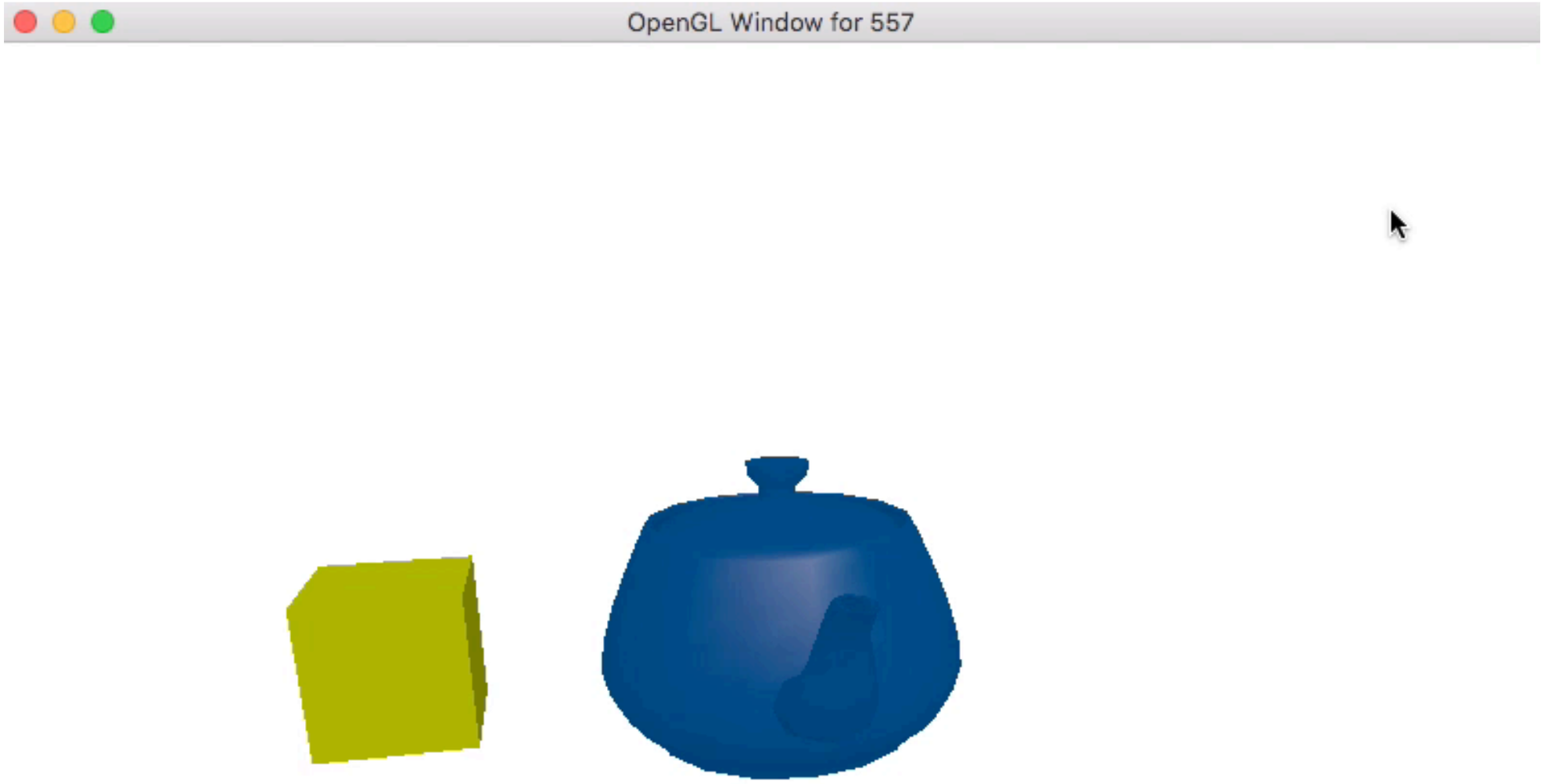
ARLAB

```
/*!
@brief - indicate that this object has been selected.
@param shader_program - the number of the glsl shader program
@param value - set is_selected true or false
*/
void setSelectColor(int shader_program,  bool value)
{
    glUseProgram(shader_program);
    int uniform_id = glGetUniformLocation(shader_program, "is_selected");

    glUniform1i(uniform_id, value);

    glUseProgram(0);
}
```

Video



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