Computação Gráfica

Computer Graphics

Engenharia Informática (11569) – 3° ano, 2° semestre

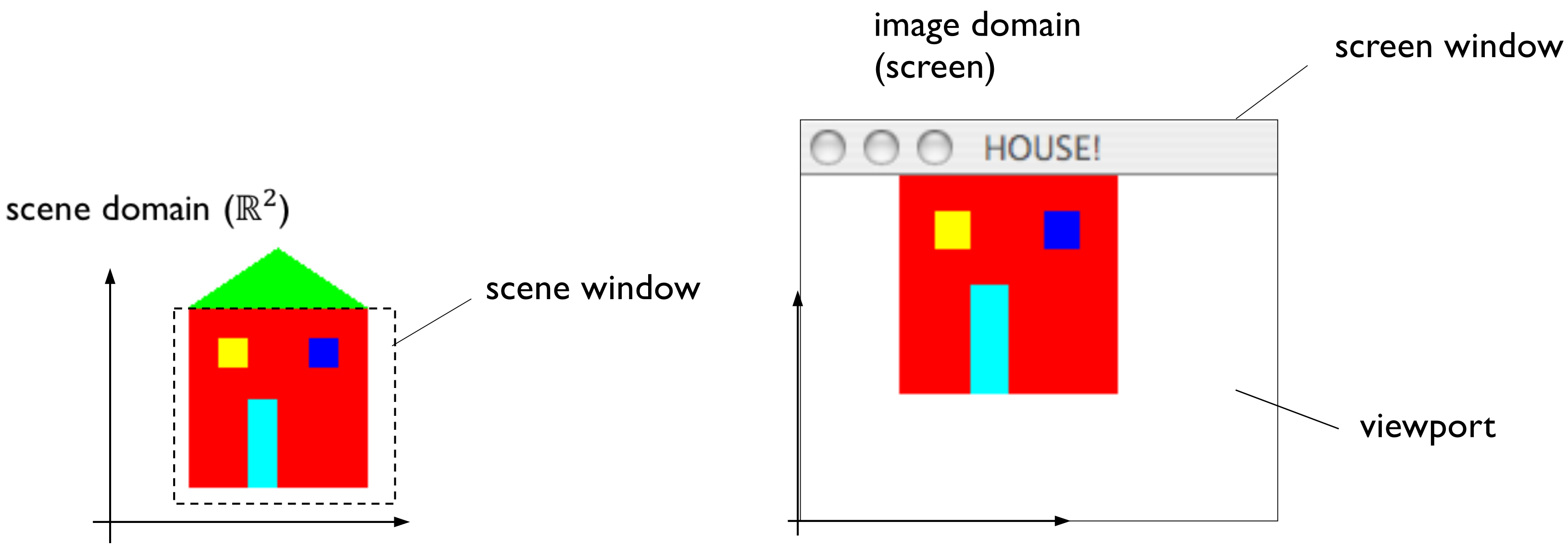
Chap. 4 – Windows and Viewports

Outline

...:

- Basic definitions in 2D:
  - Global coordinates (scene domain): continuous domain
  - Screen coordinates (image domain): discrete domain
- Window-viewport transformation
- Window-viewport transformation in OpenGL
- Geometric transformations in OpenGL/GLM
- Graphics pipeline (or rendering pipeline)
- Mapping a scene into various viewports
- Avoiding image distortion
- OpenGL examples

Definitions



Global (or world) coordinate system (or scene domain)

– is associated with the scene domain (or application domain)

– is where geometric objects lie in

– is where the geometry of each scene's object is defined

– the scene domain is <u>continuous</u> (e.g., $\mathbb{R}^2$)

Scene window (scene sub-domain)

– rectangualr scene sub-domain whose contents we intend to display on screen

Image coordinate system (or image domain)

– is associated with a screen window

–screen space where the raster image is displayed (e.g., 500x500 pixels)

– a screen window may comprise various viewports, which may overlap

– the image domain is <u>discrete</u> (pixels)

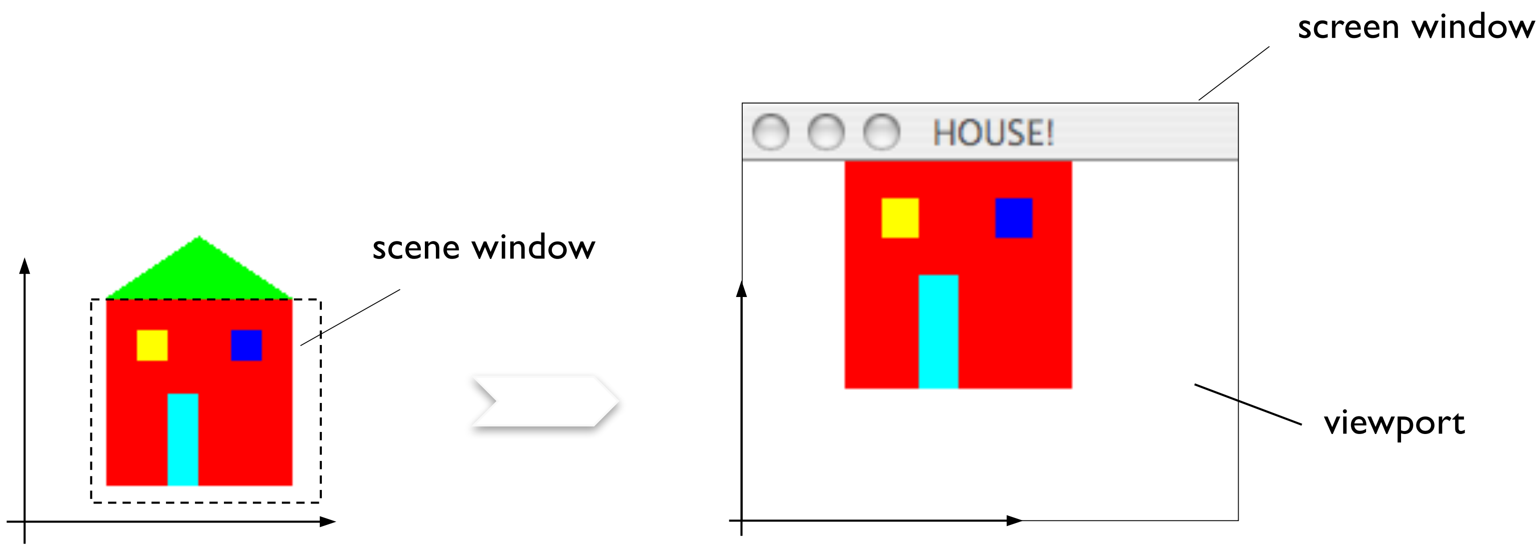
Viewport (image sub-domain)

– part of the screen window where the image is rendered

Window-viewport transformation

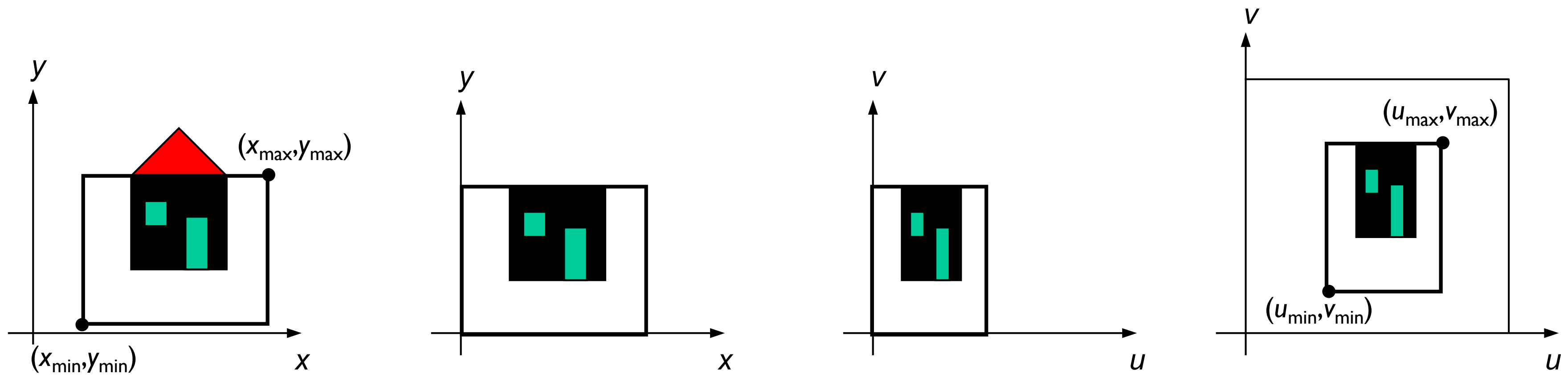
We need an automated process:

- To map a scene window (world coordinates) into a viewport (screen coordinates), the so-called window-viewport transformation.
- Thus, in principle, the same scene can be mapped into different viewports, no matter they belong to the same screen window or not.



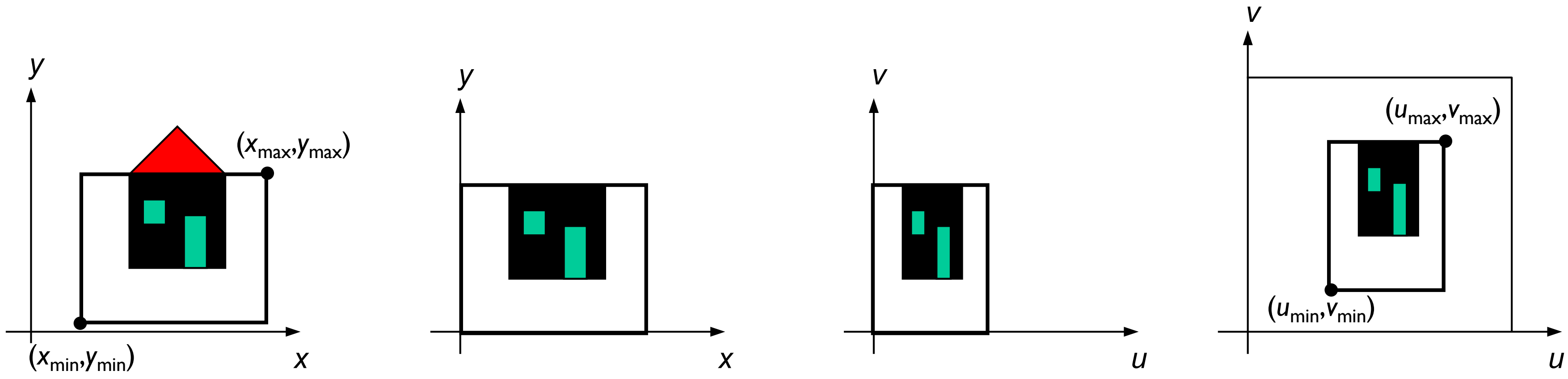
Window-viewport transformation

- *This operation is performed in an automated manner by the graphics system.*
- *It involves 3 geometric transformations: translation in global coordinates, scaling, and translation in screen coordinates.*



window in global or world coordinates (scene domain)	window translated to the origin	viewport in image coordinates	viewport translated to its pre-defined position
	$T(-x_{min}, -y_{min})$	$S\left(\frac{u_{max}-u_{min}}{x_{max}-x_{min}}, \frac{v_{max}-v_{min}}{y_{max}-y_{min}}\right)$	$T(u_{min}, v_{min})$

Window-viewport transformation: matrix representation

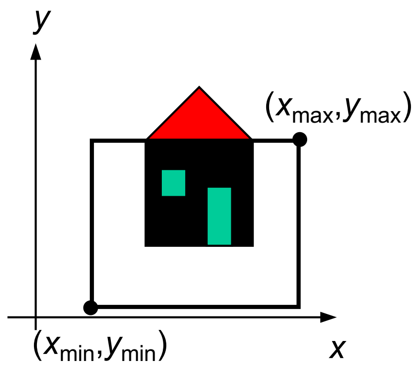


$T(-x_{min},-y_{min})$ $S(\frac{u_{max}-u_{min}}{x_{max}-x_{min}},\frac{v_{max}-v_{min}}{y_{max}-y_{min}})$ $T(u_{min},v_{min})$

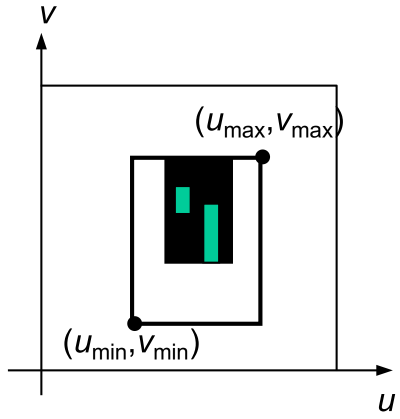
$$M_{wv} = T(u_{min},v_{min})S(\frac{u_{max}-u_{min}}{x_{max}-x_{min}},\frac{v_{max}-v_{min}}{y_{max}-y_{min}})T(-x_{min},-y_{min})$$

$$= \begin{bmatrix} 1 & 0 & u_{min} \\ 0 & 1 & v_{min} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \frac{u_{max}-u_{min}}{x_{max}-x_{min}} & 0 & 0 \\ 0 & \frac{v_{max}-v_{min}}{y_{max}-y_{min}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_{min} \\ 0 & 1 & -y_{min} \\ 0 & 0 & 1 \end{bmatrix}$$

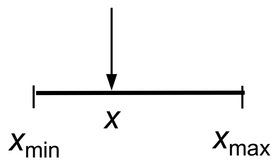
Window-viewport transformation: in more detail



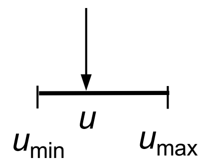
Keeping the ratio in mapping (x,y) to (u,v)

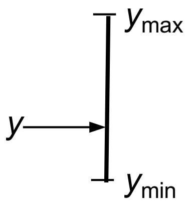


translation *scaling* *translation*

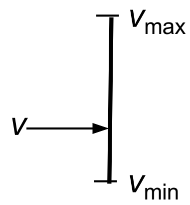


$$\frac{x - x_{min}}{x_{max} - x_{min}} = \frac{u - u_{min}}{u_{max} - u_{min}} \Leftrightarrow u = (x - x_{min}).\frac{u_{max} - u_{min}}{x_{max} - x_{min}} + u_{min}$$

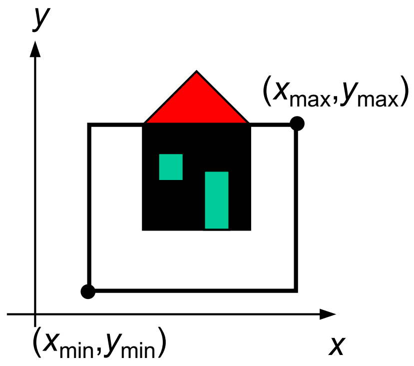


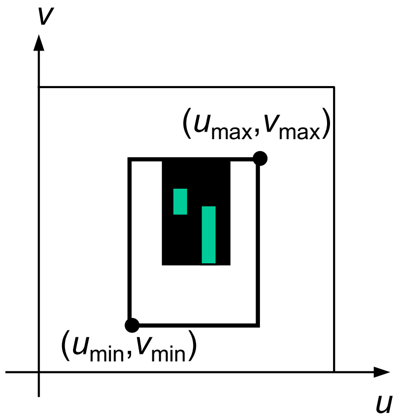


$$\frac{y - y_{min}}{y_{max} - y_{min}} = \frac{v - v_{min}}{v_{max} - v_{min}} \Leftrightarrow v = (y - y_{min}).\frac{v_{max} - v_{min}}{y_{max} - y_{min}} + v_{min}$$



Window-viewport transformation: example





window(10.0,5.0,40.0,30.0)

viewport(100,50,250,300)

$$u = (x - 10.0).\frac{250 - 100}{40.0 - 10.0} + 100 \qquad \lambda_x = \frac{250 - 100}{40.0 - 10.0} = 5.0$$

$$v = (y - 5.0).\frac{300 - 50}{30.0 - 5.0} + 50 \qquad \lambda_y = \frac{300 - 50}{30.0 - 5.0} = 10.0$$

Window-viewport transformation: GLM and OpenGL

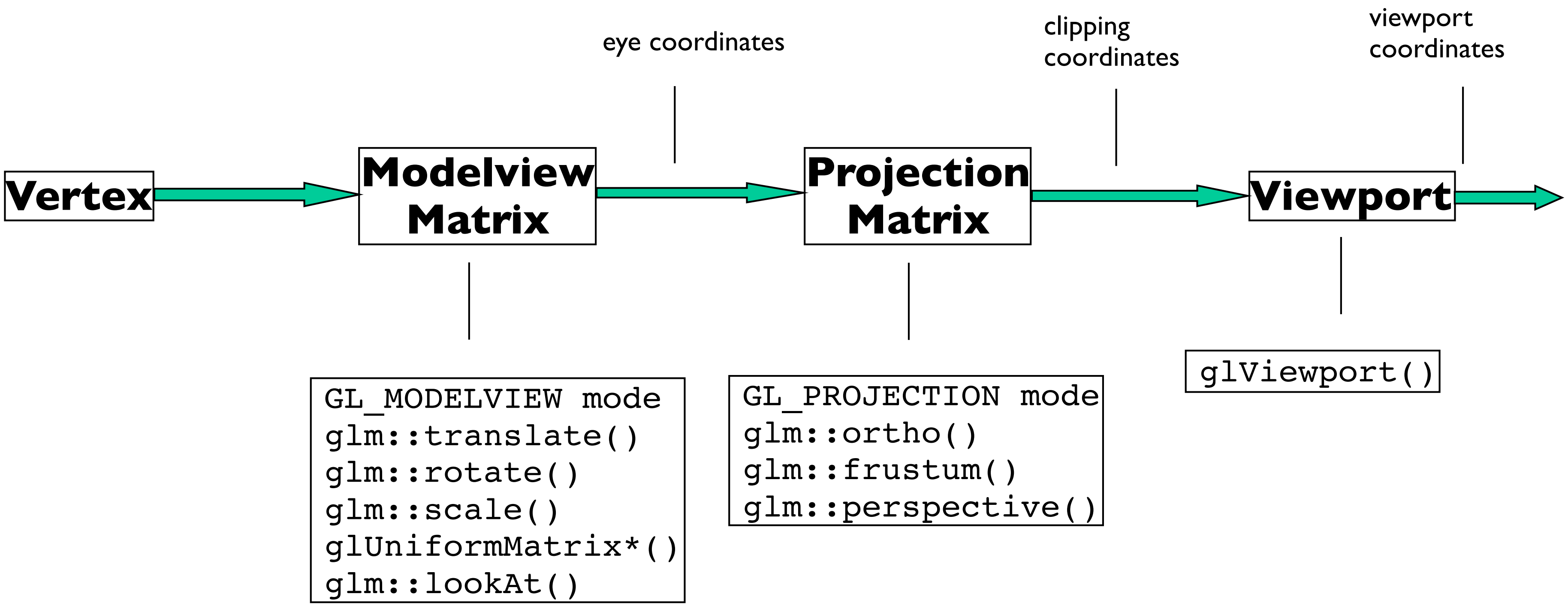
glm::**ortho**(left, right, bottom, top)

- It sets up the 2D scene domain. It is defined by two vertical lines (left and right) and two horizontal lines (bottom and top)
- Default scene domain is (-1,1,-1,1).
- It defines the orthogonal projection matrix in 2D.
- Also, it defines the window-viewport transformation in an automated manner.

glViewport(x, y, width, height)

- It defines the viewport in the screen window, where (x,y) represent its bottom-left corner, while (width,height) its size.
- By default, the viewport spans the entire domain of screen window.
- There may exist various viewports inside the screen window, which may eventually overlap.

OpenGL graphics pipeline



Example in OpenGL

(using a scene domain and a default viewport)

- In this example, we do not use the default scene domain (-1,1,-1,1); that is, the scene domain is given by glm::**ortho**(xmin,xmax,ymin,ymax)
- The function **glViewport**(x, y, width, height) ***is not*** used explicitly, so the default viewport spans the entire extent of the screen window.

P01, Exercise 3:
Graphics application to draw a house (3 triangles)

```
// Include standard headers
#include <stdio.h>
#include <stdlib.h>

// Include GLEW
#include <GL/glew.h>

// Include GLFW
#include <GLFW/glfw3.h>
GLFWwindow* window;

// GLM header file
#include <glm/glm.hpp>
using namespace glm;

// shaders header file
#include <common/shader.hpp>

// Vertex array object (VAO)
GLuint VertexArrayID;

// Vertex buffer object (VBO)
GLuint vertexbuffer;

// color buffer object (CBO)
GLuint colorbuffer;

// GLSL program from the shaders
GLuint programID;
```

```
// function prototypes
void transferDataToGPUMemory(void);
void cleanupDataFromGPU();
void draw();
```

P01, Exercise 3:
Graphics application to draw a house (3 triangles)

```
int main( void )
{
        // Initialise GLFW
        glfwInit();
        // Setting up OpenGL version and the like
        glfwWindowHint(GLFW_SAMPLES, 4);
        glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
        glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
        glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE); // To make MacOS happy; should not be needed
        glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
        // Open a window
        window = glfwCreateWindow( 1024, 768, ”House in Red and Green", NULL, NULL);
        // Create window context
        glfwMakeContextCurrent(window);
        // Initialize GLEW
        glewExperimental = true; // Needed for core profile
        glewInit();
        // Ensure we can capture the escape key being pressed below
        glfwSetInputMode(window, GLFW_STICKY_KEYS, GL_TRUE);
        // Dark blue background
        glClearColor(0.0f, 0.0f, 0.4f, 0.0f);
        // transfer my data (vertices, colors, and shaders) to GPU side
        transferDataToGPUMemory();
        // render scene for each frame
        do{     // drawing callback
                draw();
                // Swap buffers
                glfwSwapBuffers(window);
                // looking for input events
                glfwPollEvents();
        } while (glfwGetKey(window, GLFW_KEY_ESCAPE ) != GLFW_PRESS && glfwWindowShouldClose(window) == 0 );
        // Cleanup VAO, VBOs, and shaders from GPU
        cleanupDataFromGPU();
        // Close OpenGL window and terminate GLFW
        glfwTerminate();
        return 0;
}
```

P01, Exercise 3:
Graphics application to draw a house (3 triangles)

```
void transferDataToGPUMemory(void)
{
	// VAO
	glGenVertexArrays(1, &VertexArrayID);
	glBindVertexArray(VertexArrayID);

	// Create and compile our GLSL program from the shaders
	programID = LoadShaders( "SimpleVertexShader.vertexshader", "SimpleFragmentShader.fragmentshader" );
	// vertices for 2 triangles
	static const GLfloat g_vertex_buffer_data[] = {
		0.0f,  0.0f,  0.0f, 20.0f, 0.0f,  0.0f, 20.0f, 20.0f, 0.0f,    // first triangle
		0.0f,  0.0f,  0.0f, 20.0f, 20.0f, 0.0f, 0.0f,  20.0f, 0.0f,    // second triangle
		0.0f,  20.0f, 0.0f, 20.0f, 20.0f, 0.0f, 10.0f, 30.0f, 0.0f,    // third triangle
	};

	// One color for each vertex
	static const GLfloat g_color_buffer_data[] = {
		1.0f,  0.0f,  0.0f, 1.0f,  0.0f,  0.0f, 1.0f,  0.0f,  0.0f,    // color for 3 vertices of the first triangle
		1.0f,  0.0f,  0.0f, 1.0f,  0.0f,  0.0f, 1.0f,  0.0f,  0.0f,    // color for 3 vertices of the second triangle
		0.0f,  1.0f,  0.0f, 0.0f,  1.0f,  0.0f, 0.0f,  1.0f,  0.0f,    // color for 3 vertices of the third triangle
	};

	// Move vertex data to video memory; specifically to VBO called vertexbuffer
	glGenBuffers(1, &vertexbuffer);
	glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer);
	glBufferData(GL_ARRAY_BUFFER, sizeof(g_vertex_buffer_data), g_vertex_buffer_data, GL_STATIC_DRAW);
	// Move color data to video memory; specifically to CBO called colorbuffer
	glGenBuffers(1, &colorbuffer);
	glBindBuffer(GL_ARRAY_BUFFER, colorbuffer);
	glBufferData(GL_ARRAY_BUFFER, sizeof(g_color_buffer_data), g_color_buffer_data, GL_STATIC_DRAW);

}
```

P01, Exercise 3:
Graphics application to draw a house (3 triangles)

```
void cleanupDataFromGPU()
{
	glDeleteBuffers(1, &vertexbuffer);
	glDeleteBuffers(1, &colorbuffer);
	glDeleteVertexArrays(1, &VertexArrayID);
	glDeleteProgram(programID);
}
```

P01, Exercise 3:
Graphics application to draw a house (3 triangles)

```
void draw (void)
{
        // Clear the screen
        glClear( GL_COLOR_BUFFER_BIT );
        // Use our shader
        glUseProgram(programID);

        // window-viewport transformation
        glm::mat4 mvp = glm::ortho(-40.0f, 40.0f, -40.0f, 40.0f);
        // retrieve the matrix uniform locations
        unsigned int matrix = glGetUniformLocation(programID, "mvp");
        glUniformMatrix4fv(matrix, 1, GL_FALSE, &mvp[0][0]);

        // 1rst attribute buffer : vertices
        glEnableVertexAttribArray(0);
        glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer);
        glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, (void*)0);

        // 2nd attribute buffer : colors
        glEnableVertexAttribArray(1);
        glBindBuffer(GL_ARRAY_BUFFER, colorbuffer);
        glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 0, (void*)0);

        // Draw the 3 triangles !
        glDrawArrays(GL_TRIANGLES, 0, 9); // 9 indices starting at 0

        // Disable arrays of attributes for vertices
        glDisableVertexAttribArray(0);
        glDisableVertexAttribArray(1);
}
```

P01, Exercise 3:
Graphics application to draw a house (3 triangles)

vertexshader.vs

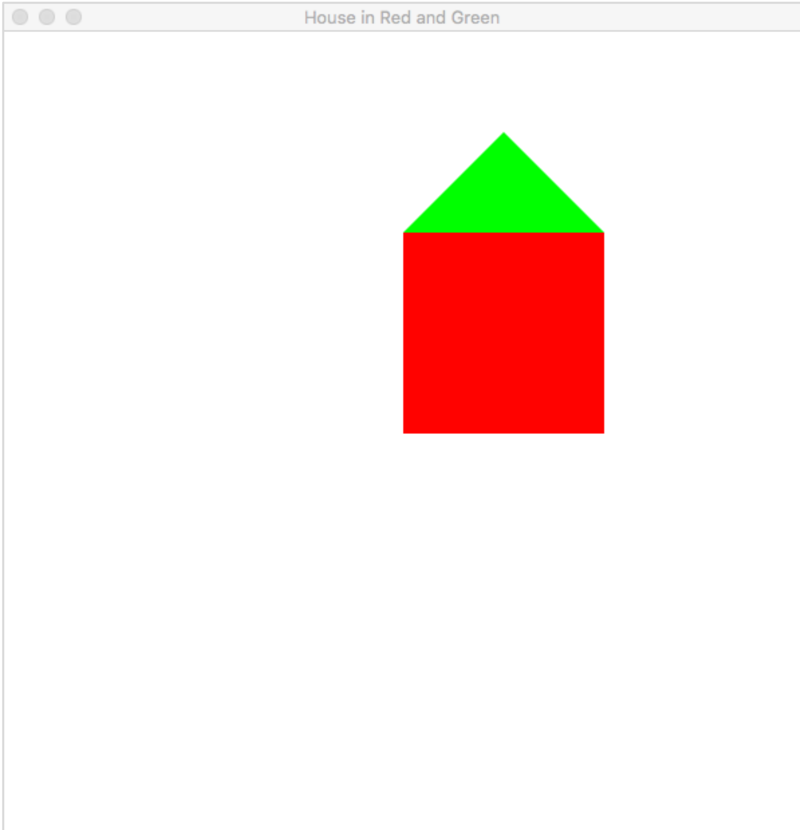
```
#version 330 core

// Input vertex data and color data
layout(location = 0) in vec3 vertexPosition;
layout(location = 1) in vec3 vertexColor;

// window-viewport transformation matrix
uniform mat4 mvp;

// Output fragment data
out vec3 fragmentColor;

void main()
{
	// project each vertex in homogeneous coordinates
	gl_Position = mvp * vec4(vertexPosition, 1.0);

	// the vertex shader just passes the color to fragment shader
	fragmentColor = vertexColor;
}
```

P01, Exercise 3:
Graphics application to draw a house (3 triangles)

fragmentshader.fs

```
#version 330 core

// Interpolated values from the vertex shaders
in vec3 fragmentColor;

// Ouput data
out vec3 color;

void main()
{
	color = fragmentColor;
}
```

P01, Exercise 3:
Graphics application to draw a house (3 triangles)



Example in OpenGL

(using a scene domain and 4 viewports)

- In this example, we use a specific scene domain given by glm::**ortho**(xmin,xmax,ymin,ymax)
- The function **glViewport**(x, y, width, height) ***is*** used explicitly to define each viewport inside the screen window.
- This program takes advantage of the same shaders as the previous program.

Graphics application to draw the same house in 4 viewports

```
// Include standard headers
#include <stdio.h>
#include <stdlib.h>

// Include GLEW
#include <GL/glew.h>

// Include GLFW
#include <GLFW/glfw3.h>
GLFWwindow* window;

// GLM header file
#include <glm/glm.hpp>
using namespace glm;

// shaders header file
#include <common/shader.hpp>

// Vertex array object (VAO)
GLuint VertexArrayID;

// Vertex buffer object (VBO)
GLuint vertexbuffer;

// color buffer object (CBO)
GLuint colorbuffer;

// GLSL program from the shaders
GLuint programID;

// screen window
GLint WindowWidth = 600;
GLint WindowHeight = 600;
```

```
// function prototypes
void transferDataToGPUMemory(void);
void cleanupDataFromGPU();
void draw();
```

Graphics application to draw the same house in 4 viewports

```
int main( void )
{         // Initialise GLFW
          glfwInit();

          // Setting up OpenGL version and the like
          glfwWindowHint(GLFW_SAMPLES, 4);
          glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
          glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
          glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE); // To make MacOS happy; should not be needed
          glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);

          // Open a window
          window = glfwCreateWindow( 1024, 768, "House in 4 Viewports", NULL, NULL);

          // Create window context
          glfwMakeContextCurrent(window);

          // Initialize GLEW
          glewExperimental = true; // Needed for core profile
          glewInit();

          // Ensure we can capture the escape key being pressed below
          glfwSetInputMode(window, GLFW_STICKY_KEYS, GL_TRUE);

          // White background
          glClearColor(1.0f, 1.0f, 1.0f, 0.0f);

          // Clear the screen
          glClear( GL_COLOR_BUFFER_BIT );

          // transfer my data (vertices, colors, and shaders) to GPU side
          transferDataToGPUMemory();

          // see next page for more …
```

Graphics application to draw the same house in 4 viewports

```
        // Create a framebuffer for viewports
        GLuint FramebufferName = 0;
        glGenFramebuffers(1, &FramebufferName);

        // render scene for each frame
        do{     //left bottom
                glBindFramebuffer(GL_FRAMEBUFFER, 0);
                glViewport(0, 0, WindowWidth*0.5, WindowHeight*0.5);
                draw();
                //right bottom
                glViewport(WindowWidth*0.5, 0, WindowWidth*0.5, WindowHeight*0.5);
                draw();
                //left top
                glViewport(0, WindowHeight*0.5, WindowWidth*0.5, WindowHeight*0.5);
                draw();
                //right top
                glViewport(WindowWidth*0.5, WindowHeight*0.5, WindowWidth*0.5, WindowHeight*0.5);
                draw();
                // Swap buffers
                glfwSwapBuffers(window);
                // looking for input events
                glfwPollEvents();
        } while (glfwGetKey(window, GLFW_KEY_ESCAPE ) != GLFW_PRESS && glfwWindowShouldClose(window) == 0 );

        // delete framebuffer
        glDeleteFramebuffers(1,&FramebufferName);

        // Cleanup VAO, VBOs, and shaders from GPU
        cleanupDataFromGPU();

        // Close OpenGL window and terminate GLFW
        glfwTerminate();
        return 0;
}
```

Graphics application to draw the same house in 4 viewports

```
void transferDataToGPUMemory(void)
{
	// VAO
	glGenVertexArrays(1, &VertexArrayID);
	glBindVertexArray(VertexArrayID);

	// Create and compile our GLSL program from the shaders
	programID = LoadShaders( "SimpleVertexShader.vertexshader", "SimpleFragmentShader.fragmentshader" );
	// vertices for 2 triangles
	static const GLfloat g_vertex_buffer_data[] = {
		0.0f,  0.0f,  0.0f, 20.0f, 0.0f,  0.0f, 20.0f, 20.0f, 0.0f,    // first triangle
		0.0f,  0.0f,  0.0f, 20.0f, 20.0f, 0.0f, 0.0f,  20.0f, 0.0f,    // second triangle
		0.0f,  20.0f, 0.0f, 20.0f, 20.0f, 0.0f, 10.0f, 30.0f, 0.0f,    // third triangle
	};

	// One color for each vertex
	static const GLfloat g_color_buffer_data[] = {
		1.0f,  0.0f,  0.0f, 1.0f,  0.0f,  0.0f, 1.0f,  0.0f,  0.0f,    // color for 3 vertices of the first triangle
		1.0f,  0.0f,  0.0f, 1.0f,  0.0f,  0.0f, 1.0f,  0.0f,  0.0f,    // color for 3 vertices of the second triangle
		0.0f,  1.0f,  0.0f, 0.0f,  1.0f,  0.0f, 0.0f,  1.0f,  0.0f,    // color for 3 vertices of the third triangle
	};

	// Move vertex data to video memory; specifically to VBO called vertexbuffer
	glGenBuffers(1, &vertexbuffer);
	glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer);
	glBufferData(GL_ARRAY_BUFFER, sizeof(g_vertex_buffer_data), g_vertex_buffer_data, GL_STATIC_DRAW);
	// Move color data to video memory; specifically to CBO called colorbuffer
	glGenBuffers(1, &colorbuffer);
	glBindBuffer(GL_ARRAY_BUFFER, colorbuffer);
	glBufferData(GL_ARRAY_BUFFER, sizeof(g_color_buffer_data), g_color_buffer_data, GL_STATIC_DRAW);

}
```

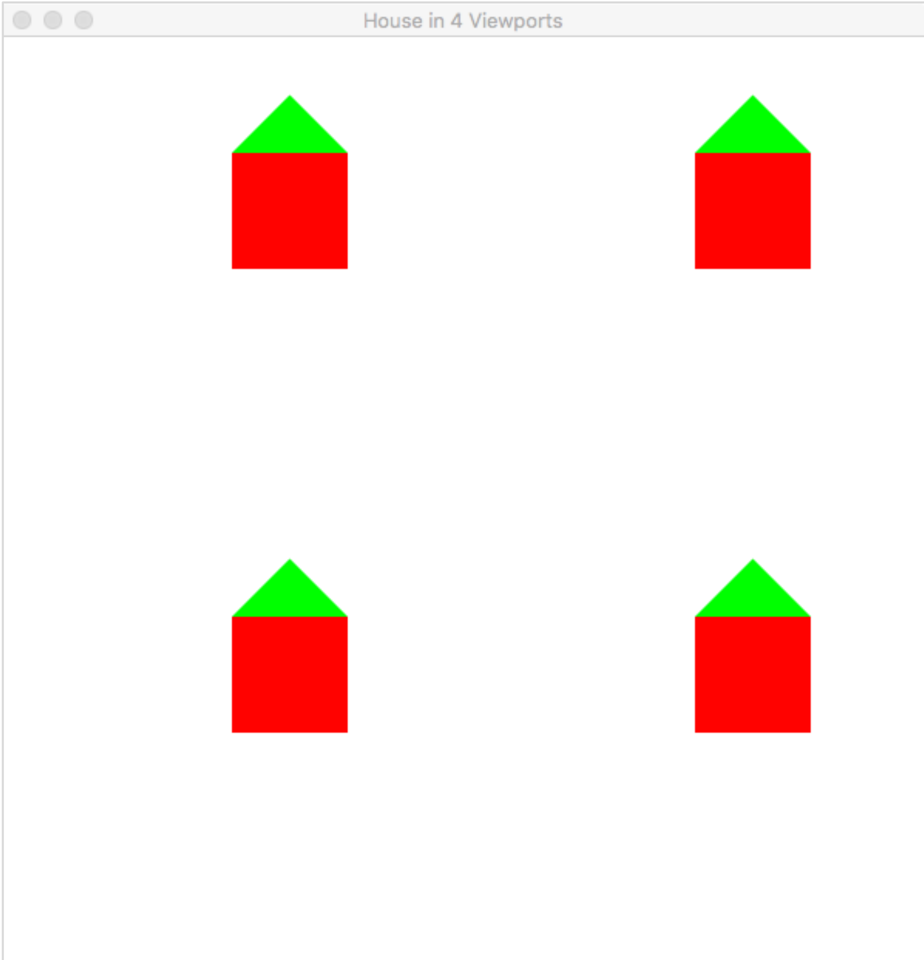
Graphics application to draw the same house in 4 viewports

```
void cleanupDataFromGPU()
{
	glDeleteBuffers(1, &vertexbuffer);
	glDeleteBuffers(1, &colorbuffer);
	glDeleteVertexArrays(1, &VertexArrayID);
	glDeleteProgram(programID);
}
```

Graphics application to draw the same house in 4 viewports

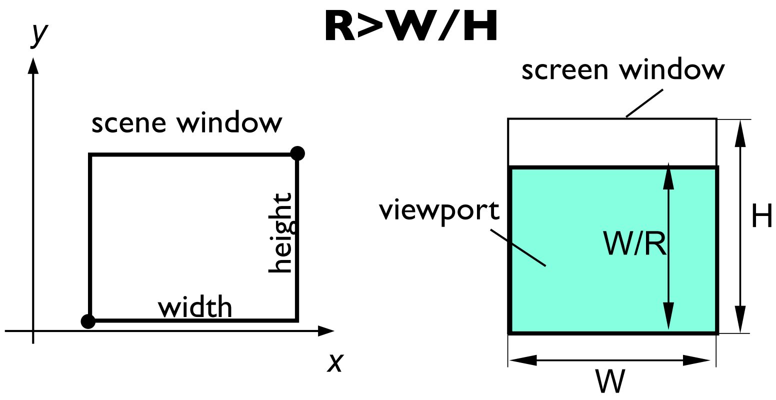
```
void draw (void)
{
	// Clear the screen : this has been moved to main(); otherwise only one house is displayed
	// glClear( GL_COLOR_BUFFER_BIT );

	// Use our shader
	glUseProgram(programID);

	// window-viewport transformation
	glm::mat4 mvp = glm::ortho(-40.0f, 40.0f, -40.0f, 40.0f);
	// retrieve the matrix uniform locations
	unsigned int matrix = glGetUniformLocation(programID, "mvp");
	glUniformMatrix4fv(matrix, 1, GL_FALSE, &mvp[0][0]);

	// 1rst attribute buffer : vertices
	glEnableVertexAttribArray(0);
	glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer);
	glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, (void*)0);

	// 2nd attribute buffer : colors
	glEnableVertexAttribArray(1);
	glBindBuffer(GL_ARRAY_BUFFER, colorbuffer);
	glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 0, (void*)0);

	// Draw the 3 triangles !
	glDrawArrays(GL_TRIANGLES, 0, 9); // 9 indices starting at 0

	// Disable arrays of attributes for vertices
	glDisableVertexAttribArray(0);
	glDisableVertexAttribArray(1);
}
```

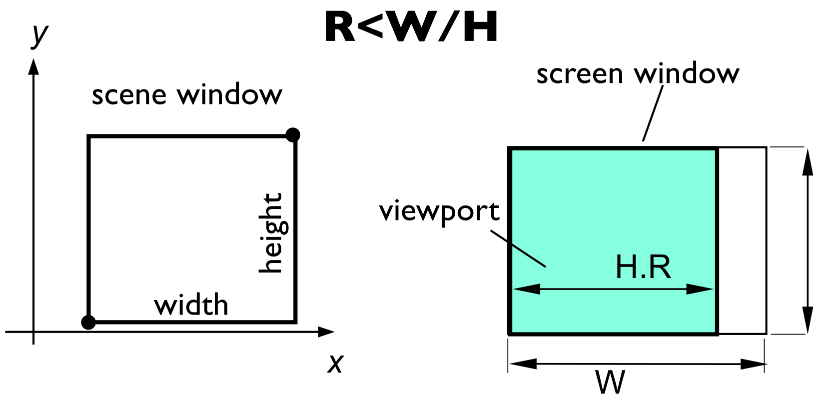
Graphics application to draw the same house in 4 viewports



Automated setup of the viewport without image distortion

- R = aspect ratio of the scene domain or window = width / height
- We consider two cases:





- To avoid image distortion, a rectangular region on the top of screen window will be discarded.
- Thus, the mapped image will own the resolution $W \times W/R$.

- To avoid image distortion, a rectangular region on the right-hand side of screen window will be discarded.
- Thus, the mapped image will own the resolution $H * R \times H$.

```
glViewport(0,0,W,W/R);
```

```
glViewport(0,0,H*R,H);
```

Summary:

. . .:

- Basic definitions in 2D:
  - Global coordinates (scene domain): continuous domain
  - Screen coordinates (image domain): discrete domain
- Window-viewport transformation
- Window-viewport transformation in OpenGL
- Geometric transformations in OpenGL/GLM
- Graphics pipeline (or rendering pipeline)
- Mapping a scene into various viewports
- Avoiding image distortion
- OpenGL examples