

ED1021 - Introduction to computation and visualisation

L13 - Introduction to Open Graphics Library (OpenGL)

Ramanathan Muthuganapathy (<https://ed.iitm.ac.in/~raman>)

Course web page: <https://ed.iitm.ac.in/~raman/introcomp.html>

Moodle page: Available at <https://courses.iitm.ac.in/>

OpenGL - what can be done?

Application of C programming

- To draw (render) primitive / objects / 3D shape / Scenes
- Lighting and shading
- Surfaces
- Animation
- Games
- OpenGL windowing system
- Independent of OS

Installation instructions

- For Windows users - a video was sent with instructions.
- Mac users -->
 - Project --> General --> Linked Libraries and Frameworks --> Add GLUT.framework and OpenGL.framework

OpenGL versions

1.x to 4.x

- Legacy OpenGL (before 3.0)
- Modern OpenGL (from 3.0)
- Current version is 4.6
- https://www.khronos.org/opengl/wiki/History_of_OpenGL#:~:text=Official%20versions%20of%20OpenGL%20released,%2C%204.4%2C%204.5%2C%204.6.
- <https://www.opengl.org/>
- We will mostly use a very basic version (mostly 1.x)

Resources

All are available on the web

- OpenGL Redbook - <https://www.glprogramming.com/red/> (search for OpenGL redbook)
- Sample programs are available at <https://www.opengl.org/archives/resources/code/samples/redbook/> and
- https://www.opengl.org/archives/resources/code/samples/glut_examples/examples/examples.html
- <http://docs.gl/> (You can go and search about a particular function)

What does OpenGL actually consists of?

- Functions
- Application Programming Interfaces (APIs)

How will you access functions?

- How do you get access to scanf()?
- Header files
 - #include <GL/glut.h> (in Mac, #include <GLUT/glut.h>)
- Usually sufficient
 - #include <GL/gl.h>
 - #include <GL/glu.h>
 - #include <GL/glut.h>

Header files

- gl.h - consists of all the rendering (drawing - model and viewing related) functions
- glu.h - contains some utility functions
- glut.h - OpenGL windows programming functions.

Demo

simple.c and hello.c

Demo

simple.c and hello.c

- CW: Try the same in your computer.
- Observations?

Observations

(name of the functions and function calls)

- some functions starts with 'gl', e.x.?
- some functions starts with 'glu', e.x.?
- some functions starts with 'glut', e.x.?

Typical order

(name of the functions and function calls)

- Create a window (and some related functions)
- Initialize the window
- Render (basically 'draw'), Resize, Input from mouse / keyboard
- Event processing loop.

Windowing related functions

```
int
main(int argc, char **argv) // command line arguments
{
    // Write your input statements before calling the window
    glutInit(&argc, argv); // Initializing the window
    glutInitWindowSize(500, 500); // Add this one
    glutCreateWindow("Simple Triangle"); // Creating the window
    glutDisplayFunc(display); // Call back function for display
    glutReshapeFunc(reshape); // When window is resized
    glutMainLoop();
    return 0;                /* ANSI C requires main to return int. */
}
```

Locate the default origin

Comment out the glutReshapeFunc()

```
void
display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glPointSize(10.0);
    glColor3f(1.0, 0.0, 0.0);  /* blue */
    glBegin(GL_POINTS);
        glVertex2i(0, 0);
    glEnd();
    glFlush();  /* Single buffered, so needs a flush. */
}
```

Change coordinates in display function

```
void
display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glColor3f(0.0, 0.0, 1.0);  /* blue */
    glBegin(GL_POLYGON);
    glVertex2i(-250, 250);
    glVertex2i(250, 250);
    glVertex2i(250, -250);
    glVertex2i(-250, -250);
    glEnd();

    glFlush();  /* Single buffered, so needs a flush. */
}
```

glOrtho() and glViewport()

window and screen/viewport coordinates

```
void
reshape(int w, int h)
{
    /* Because Gil specified "screen coordinates" (presumably with an
       upper-left origin), this short bit of code sets up the coordinate
       system to correspond to actual window coordinates. This code
       wouldn't be required if you chose a (more typical in 3D) abstract
       coordinate system. */

    glViewport(0, 0, w, h);          /* Establish viewing area to cover entire window. */
    glMatrixMode(GL_PROJECTION);    /* Start modifying the projection matrix. */
    glLoadIdentity();               /* Reset project matrix. */
    glOrtho(0, w, 0, h, -1, 1);     /* Map abstract coords directly to window coords. */
}
```

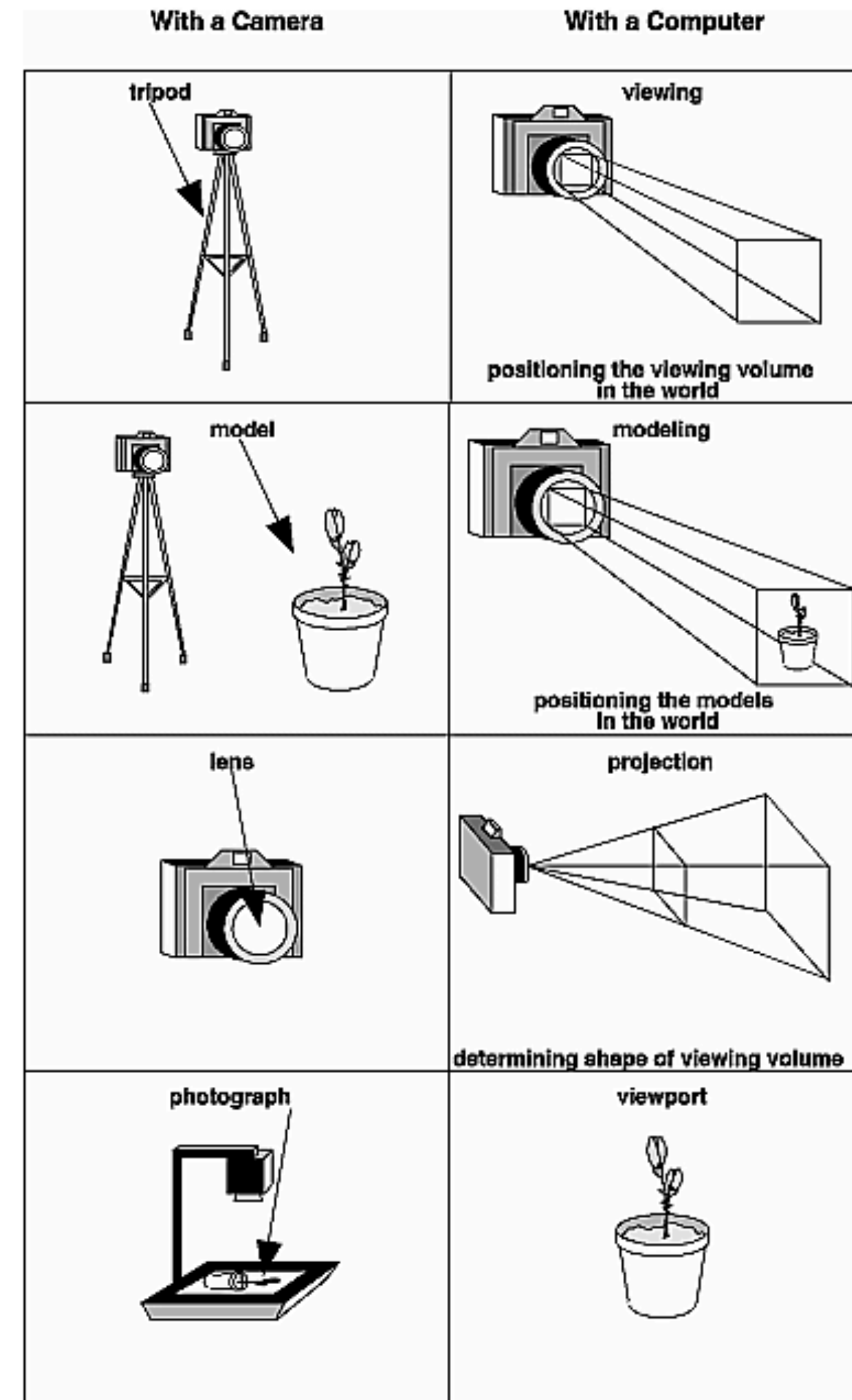

Transformations in OpenGL

Camera analogy - Fig. 3.1 in chapter 3

- Modeling
- Viewing
 - Camera orientation
 - Projection
- Map to Screen coordinates

Camera Analogy

Fig. 3.1 in chapter 3 in redbook



Coordinate systems

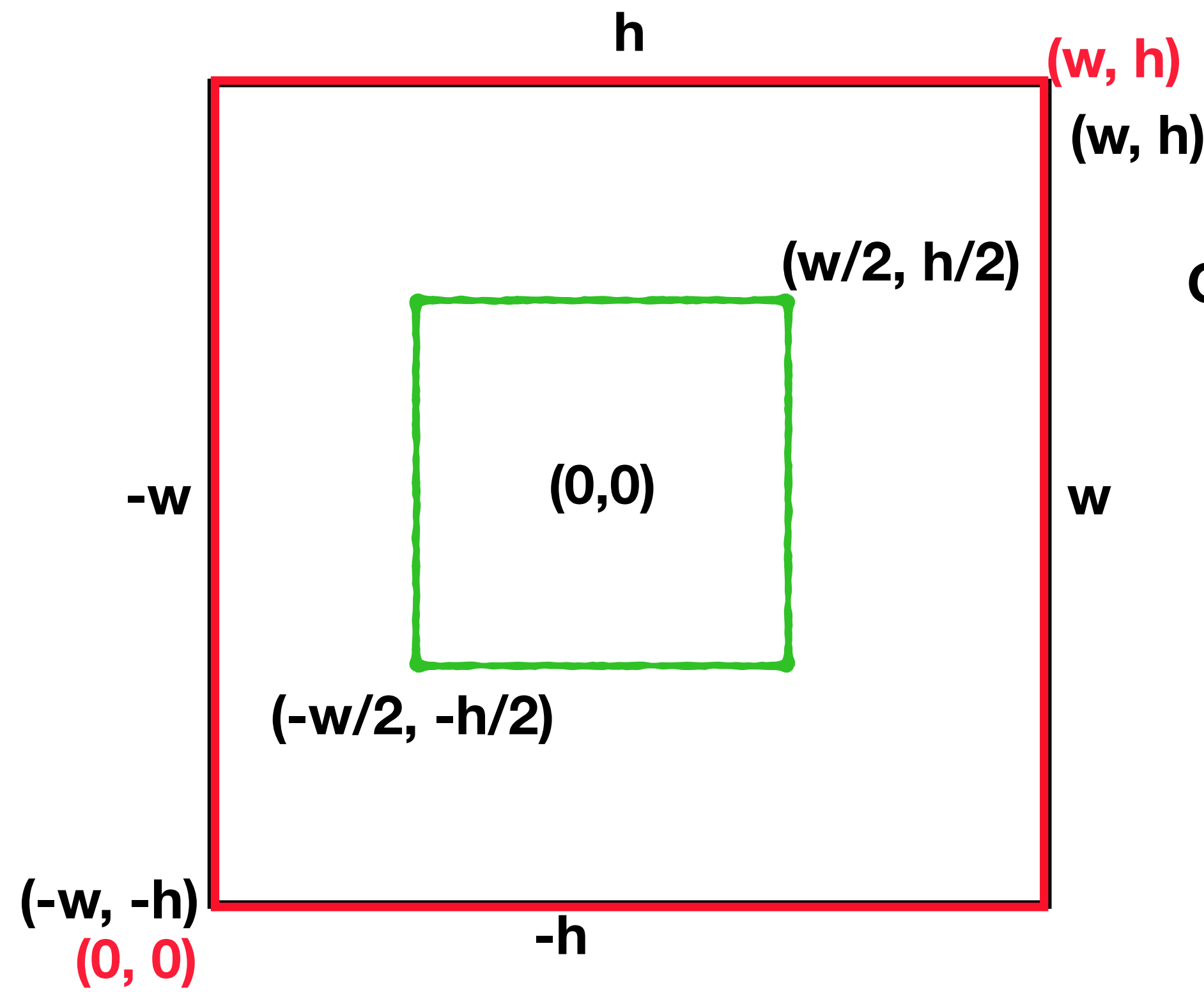
- Model (World coordinates)
- Camera (its coordinates)
- Project (Window coordinates)
- Viewport mapping (Screen coordinates)

Case 1A

Square - Object to be displayed



$(-250, 250), (-250, -250),$
 $(250, -250), (250, 250)$



Ortho $(-w, w, -h, h)$ - Black window

Viewport $(0, 0, w, h)$ - Red Window

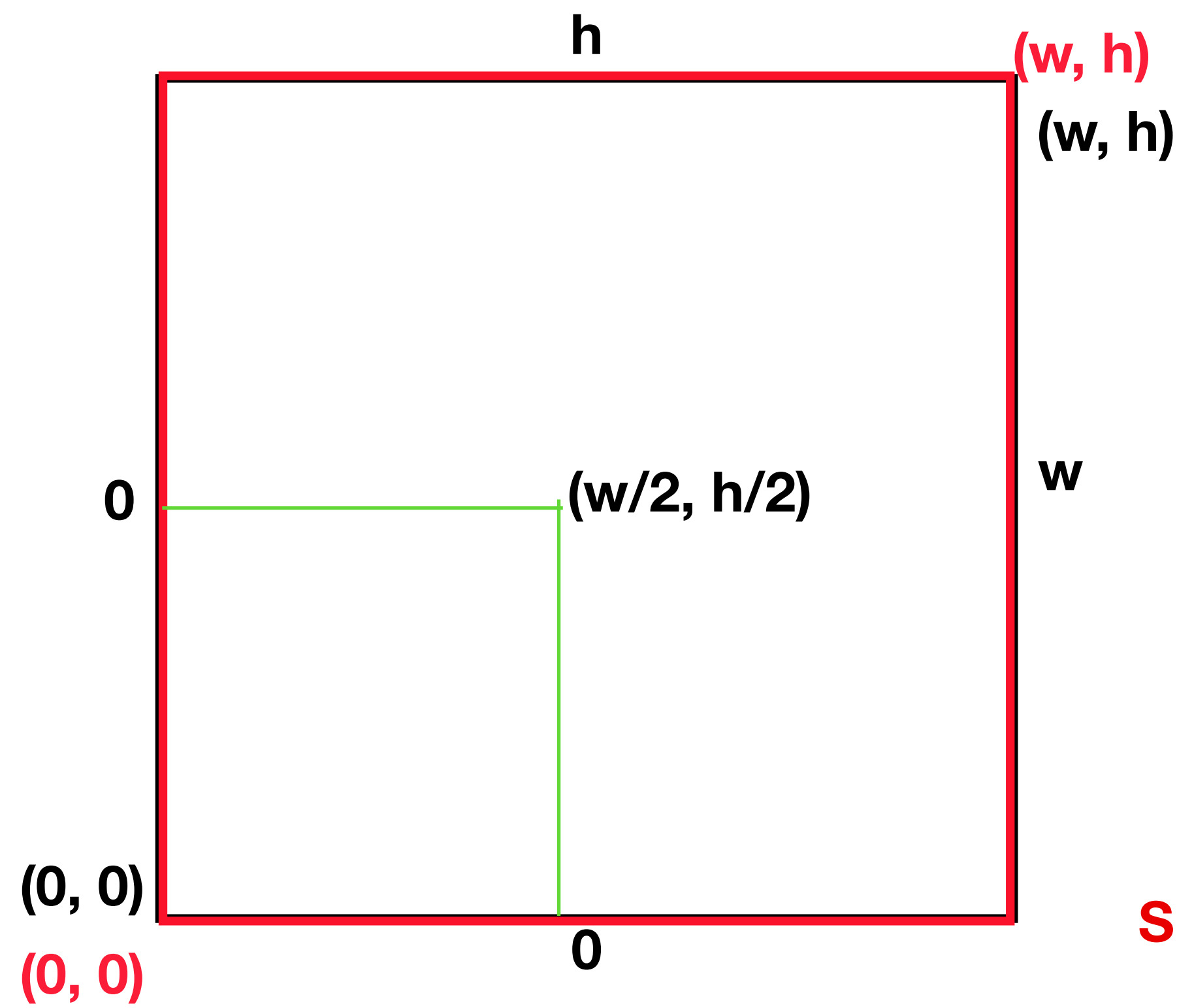
Output: Full object gets displayed

**ALL OBJECTS TO BE DISPLAYED
IN THE BLACK WINDOW**

**NOTE: In viewport, first two are the
coordinates of the lower left
But then the next two are the
SIZE of the viewport (and NOT coordinates
Of the right corner)**

Case 1B

Case 1* - Viewport(0, 0, x, y),
x and y are variables



Major portion of the square
will go out of the screen but
Bigger

Square - Object to be displayed



(-250, 250), (-250, -250),
(250, -250), (250, 250)

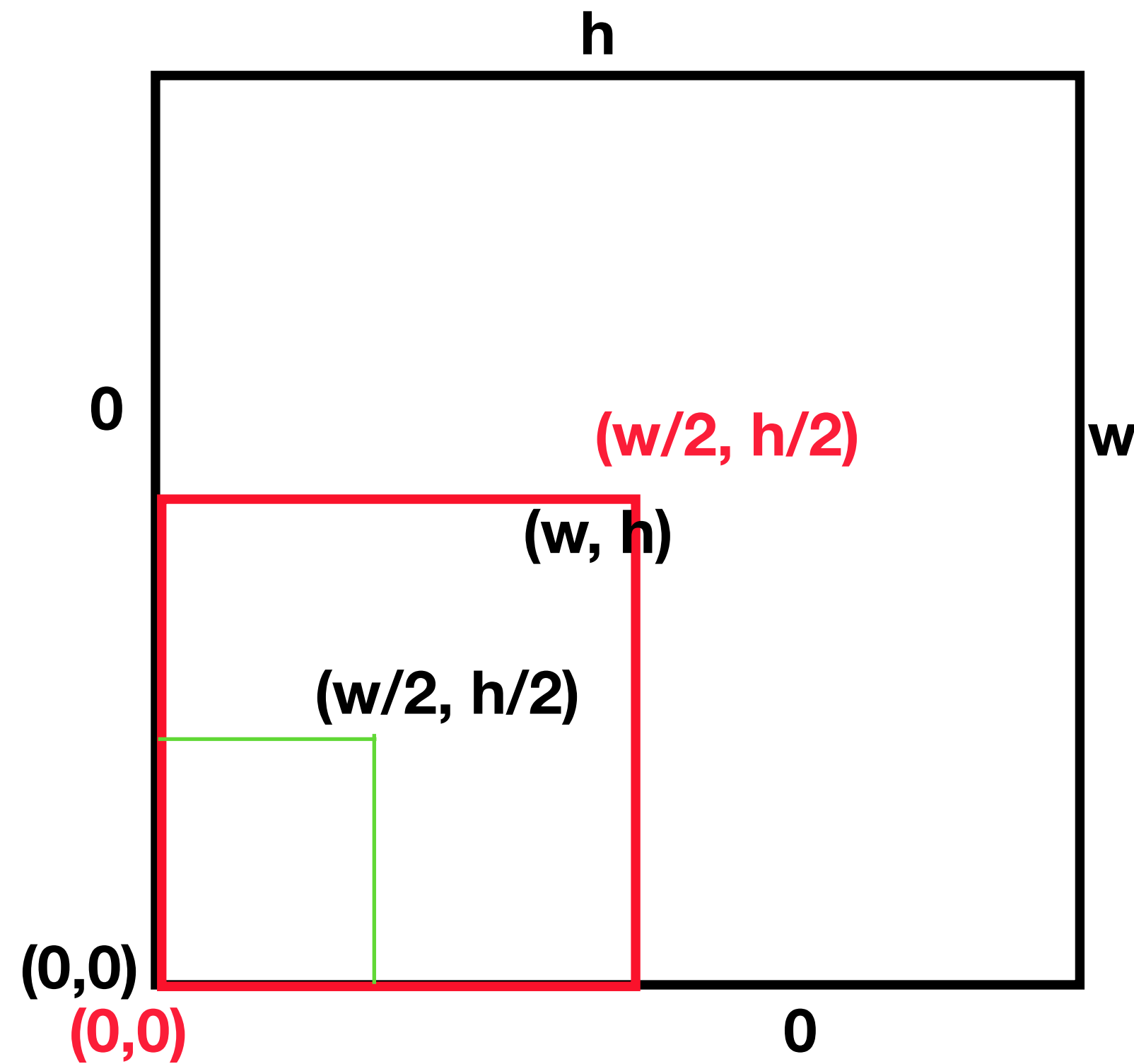
Ortho(0, w, 0, h) - Black
window

Viewport(0, 0, w, h) - Red Window

ALL OBJECTS TO BE DISPLAYED
IN THE BLACK WINDOW

NOTE: In viewport, first two are the
coordinates of the lower left
But then the next two are the
SIZE of the viewport (and NOT coordinates
Of the right corner)

Case 1C



Square - Object to be displayed

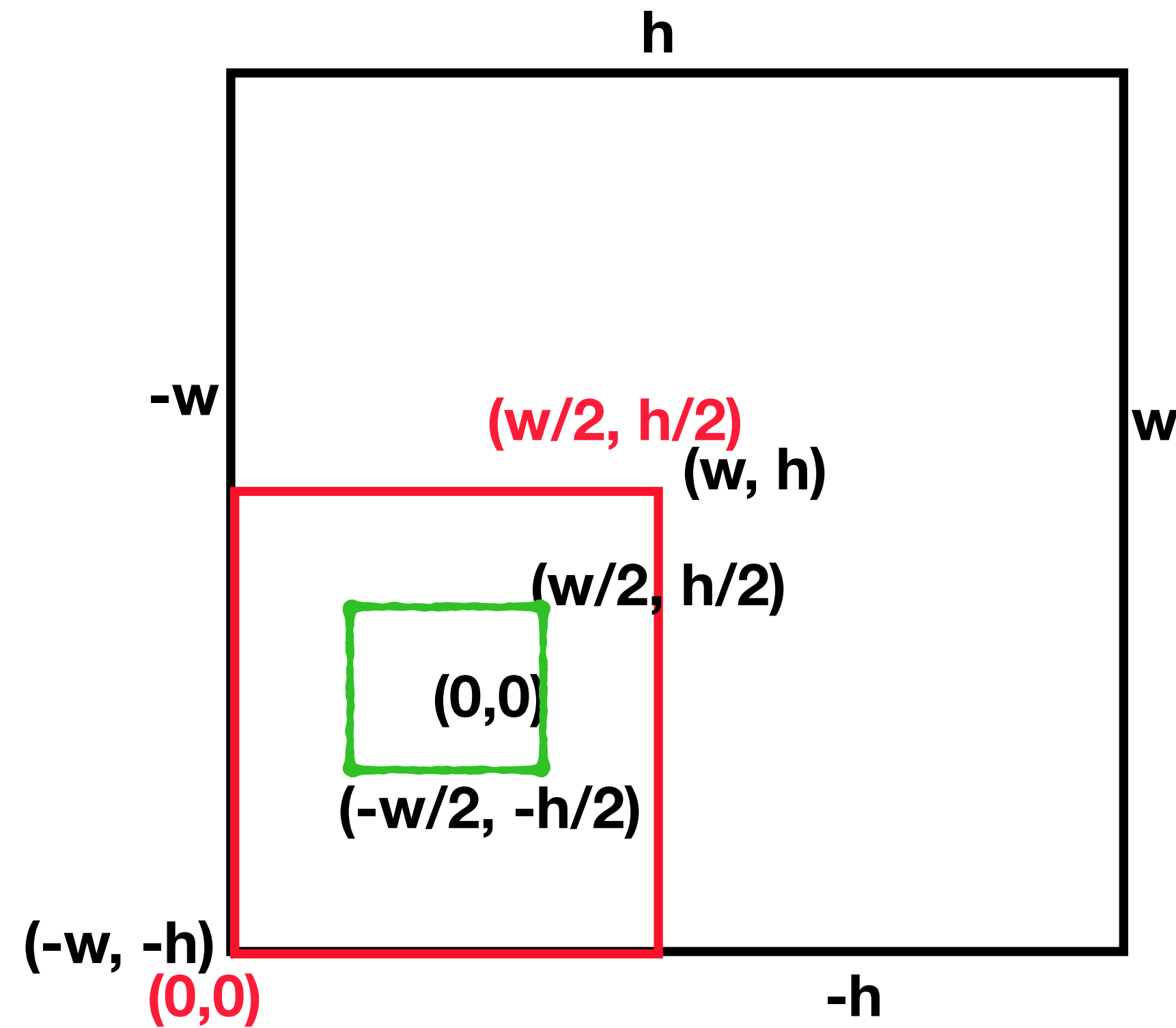
$(-250, 250), (-250, -250),$
 $(250, -250), (250, 250)$

Ortho(0, w, 0, h) - Black window
(Actual screen)

Viewport(0, 0, w/2, h/2) - Viewport

Output: Truncated, seen at
Bottom left corner of black screen
(original size)

Case 1D



Square - Object to be displayed

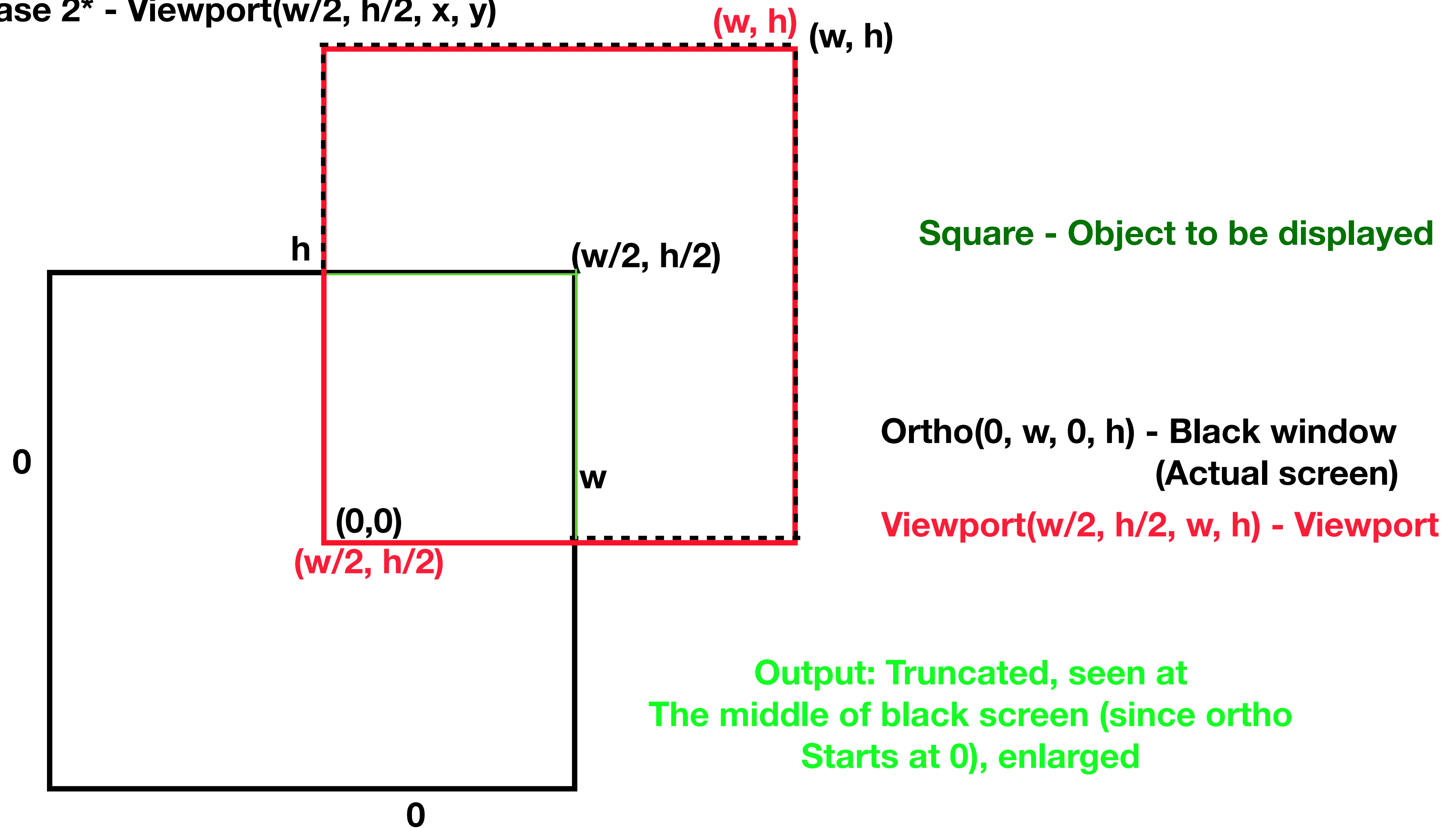
**Ortho(-w, w, -h, h) - Black window
(Actual screen)**

Viewport(0, 0, w/2, h/2) - Viewport

**Output: Fully seen at
Bottom quarter but smaller**

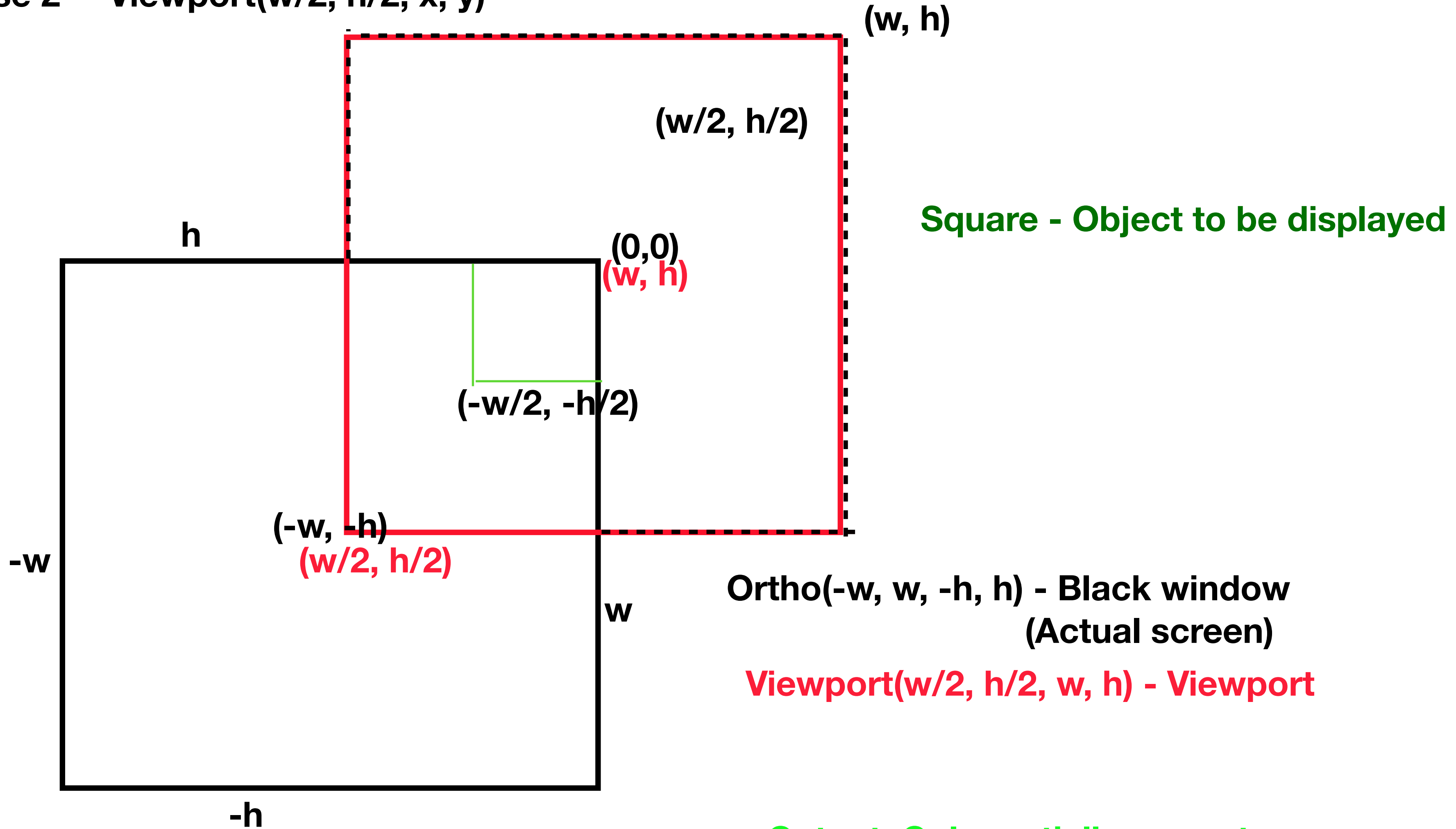
Case 2A

Case 2* - Viewport(w/2, h/2, x, y)



Case 2B

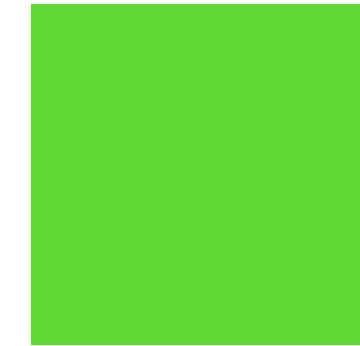
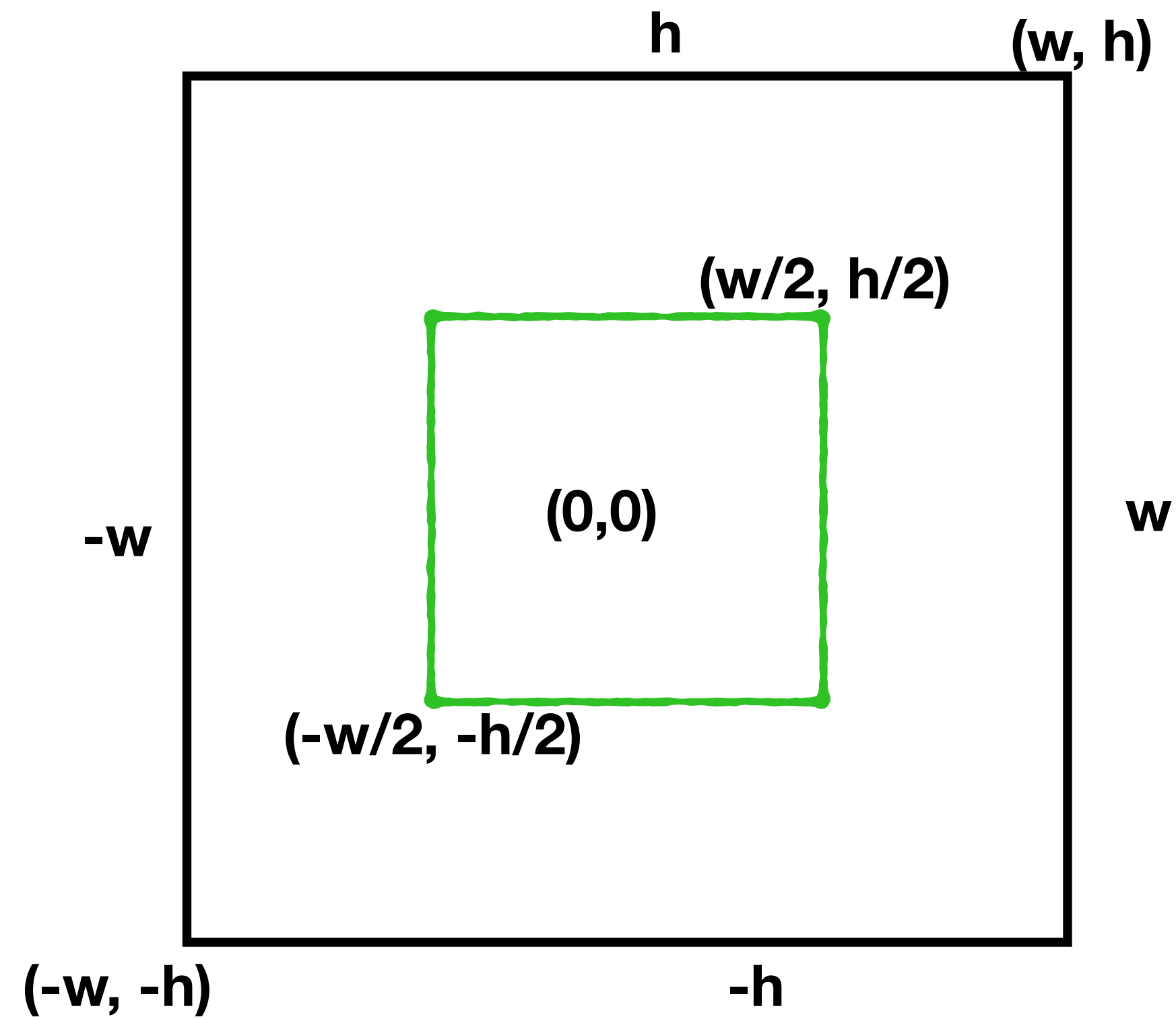
Case 2* - Viewport(w/2, h/2, x, y)



**Output: Only partially seen at
Top right corner of black screen,
Same size.**

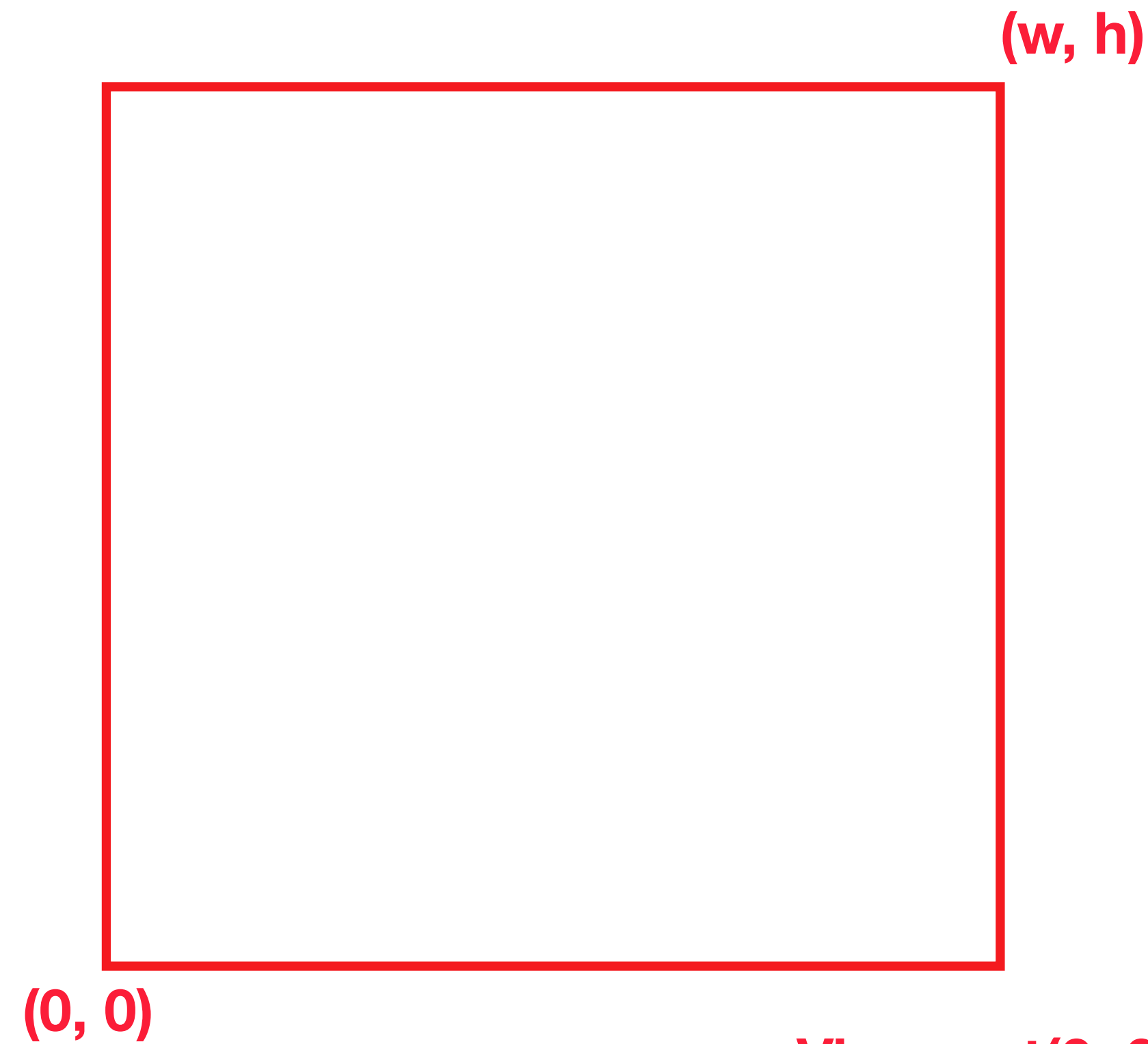
Case 2B

Ortho(-w, w, -h, h) - Black window
(Actual screen)



(-250, 250), (-250, -250),
(250, -250), (250, 250)

Square - Object to be displayed



Viewport(0, 0, w, h) - Viewport

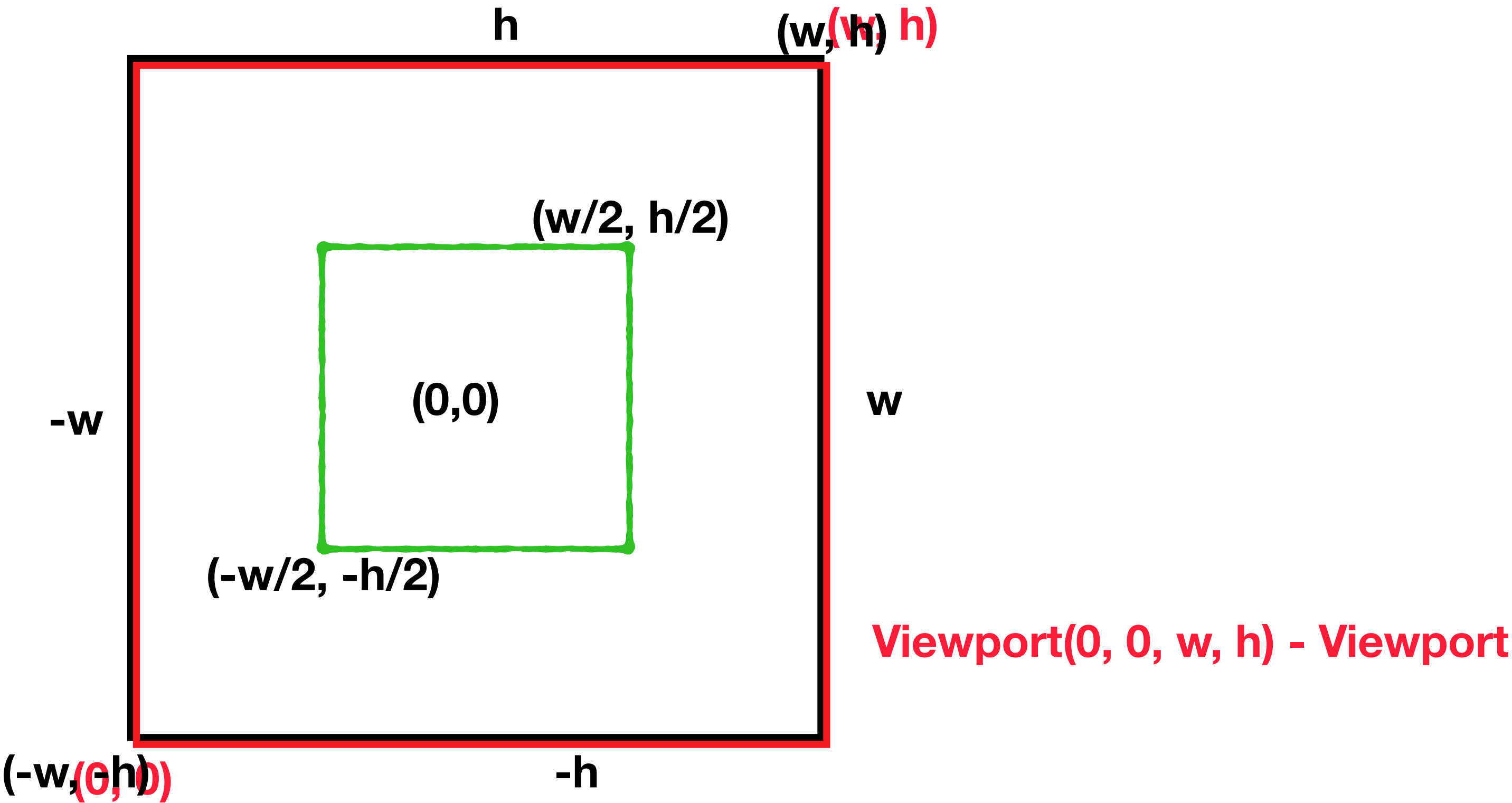
Case 2B

Ortho(-w, w, -h, h) - Black window
(Actual screen)



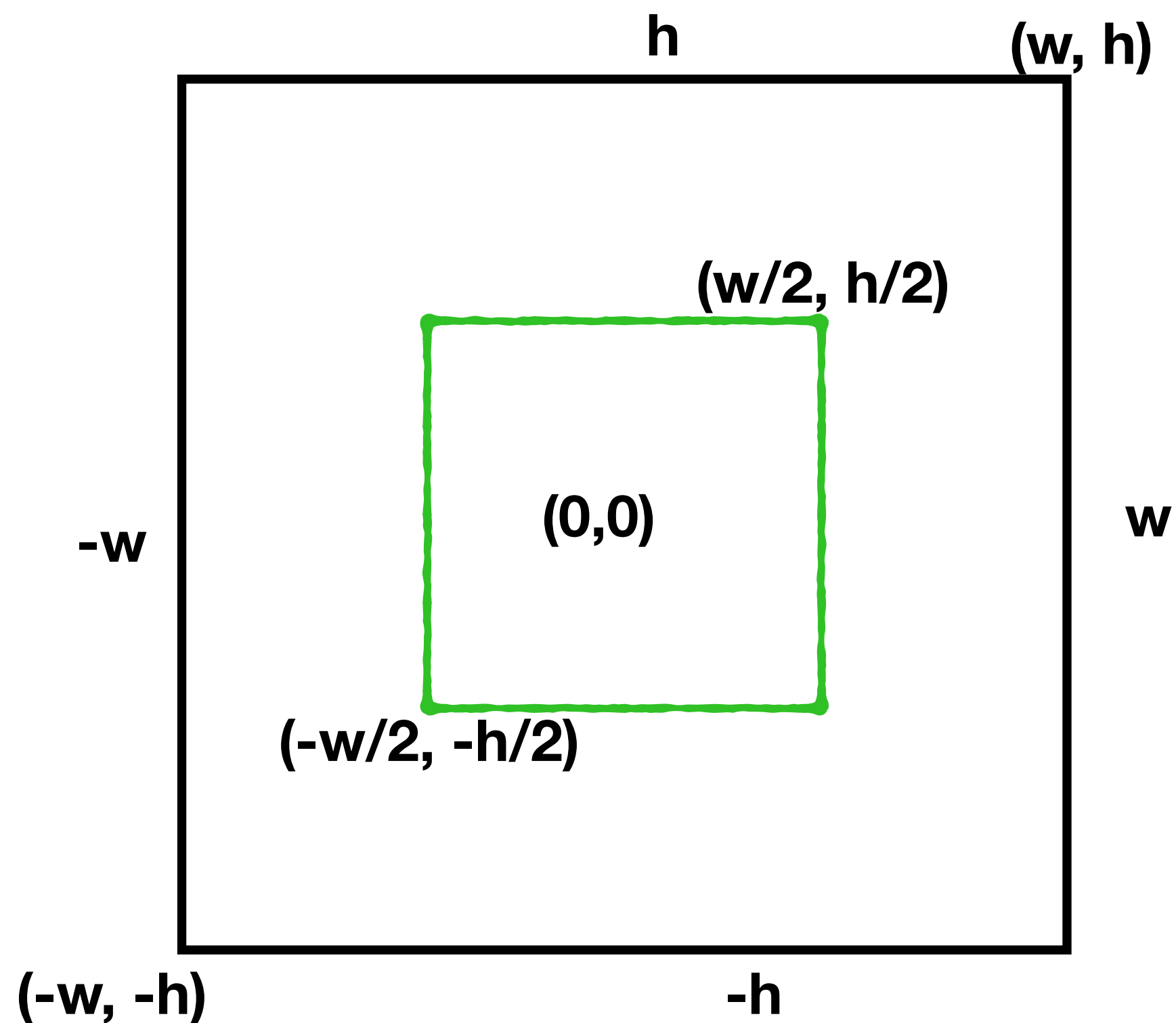
(-250, 250), (-250, -250),
(250, -250), (250, 250)

Square - Object to be displayed



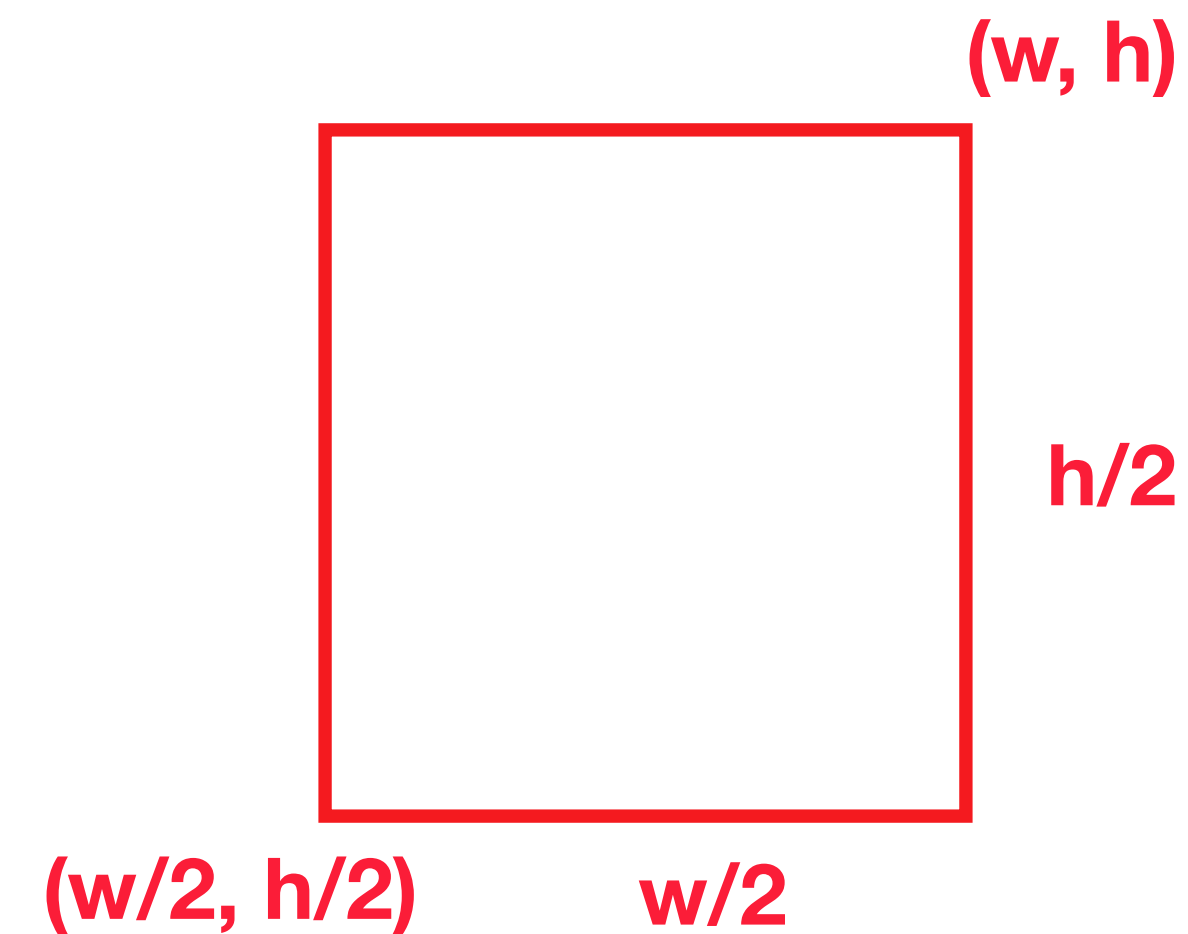
ViewportOrtho.c

Ortho(-w, w, -h, h) - Black window
(Actual screen)



(-250, 250), (-250, -250),
(250, -250), (250, 250)

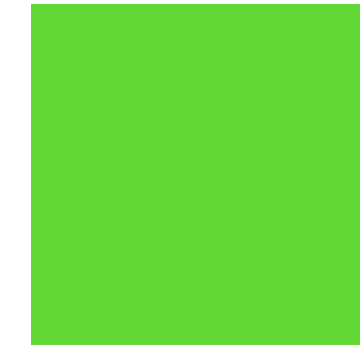
Square - Object to be displayed



Viewport(w/2, h/2, w/2, h/2) - Viewport

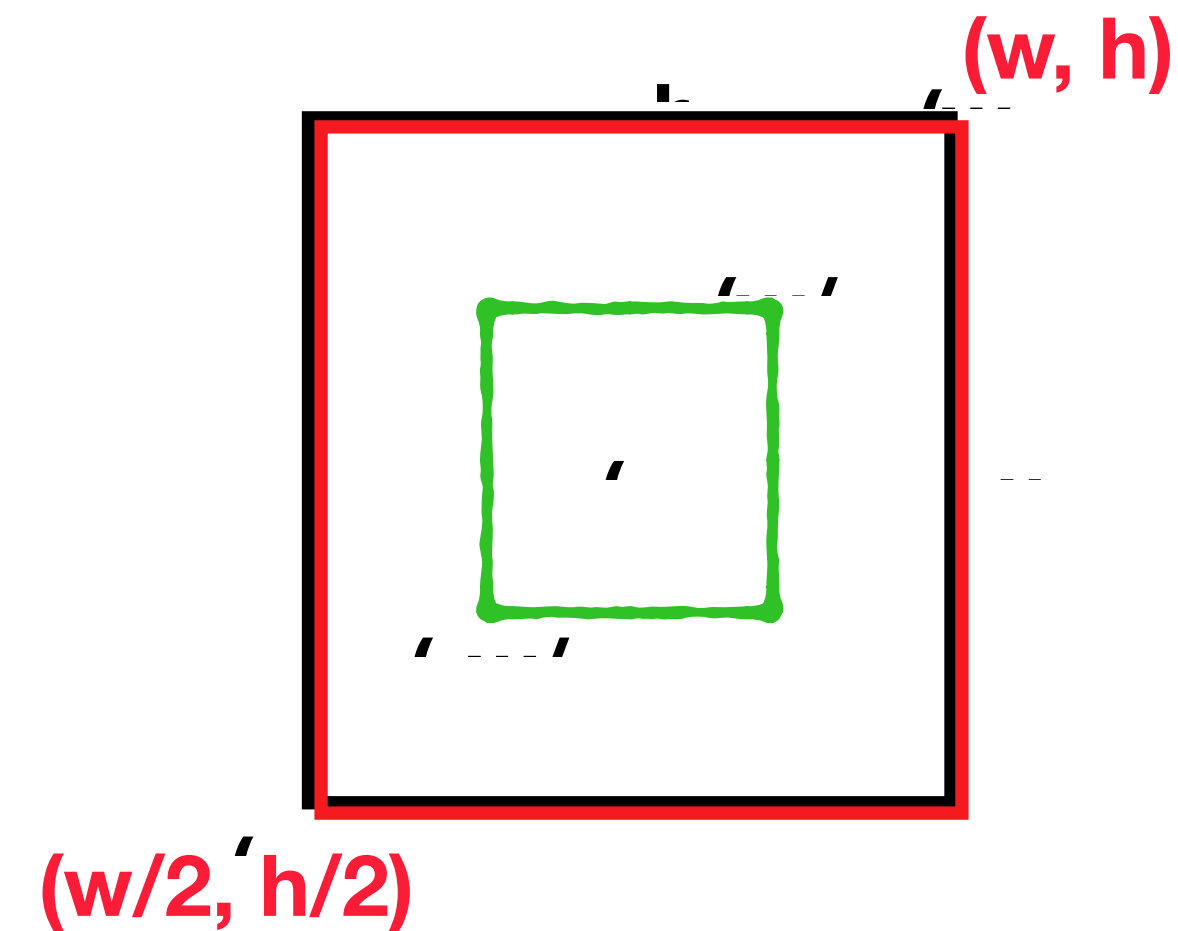
ViewportOrtho.c

Ortho(-w, w, -h, h) - Black window
(Actual screen)



$(-250, 250), (-250, -250),$
 $(250, -250), (250, 250)$

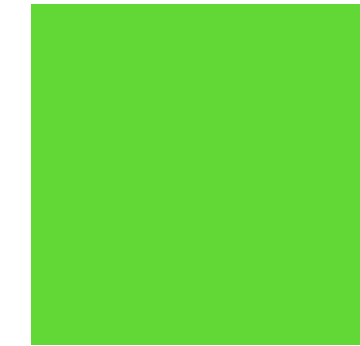
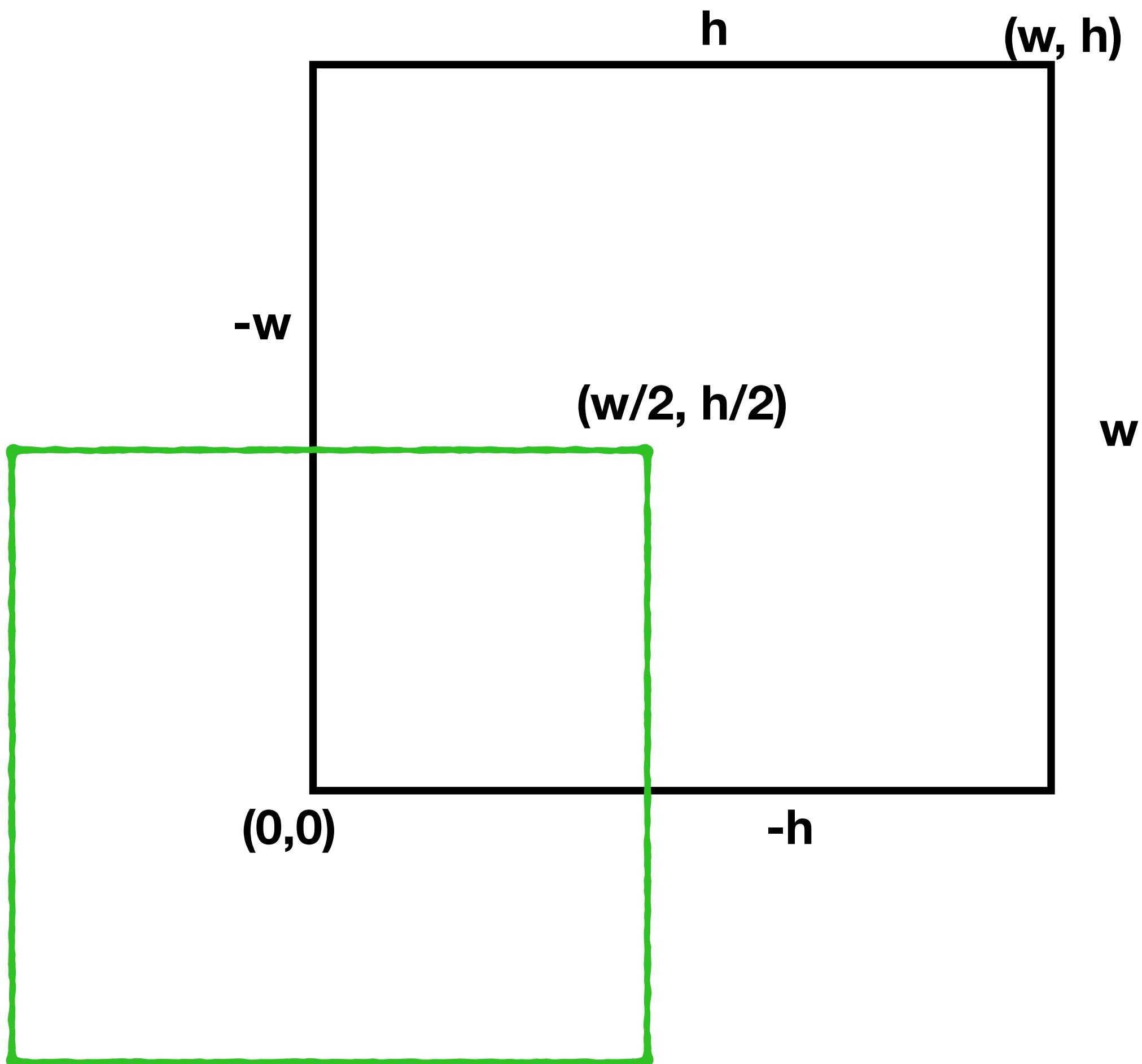
Square - Object to be displayed



Viewport($w/2, h/2, w/2, h/2$) - Viewport

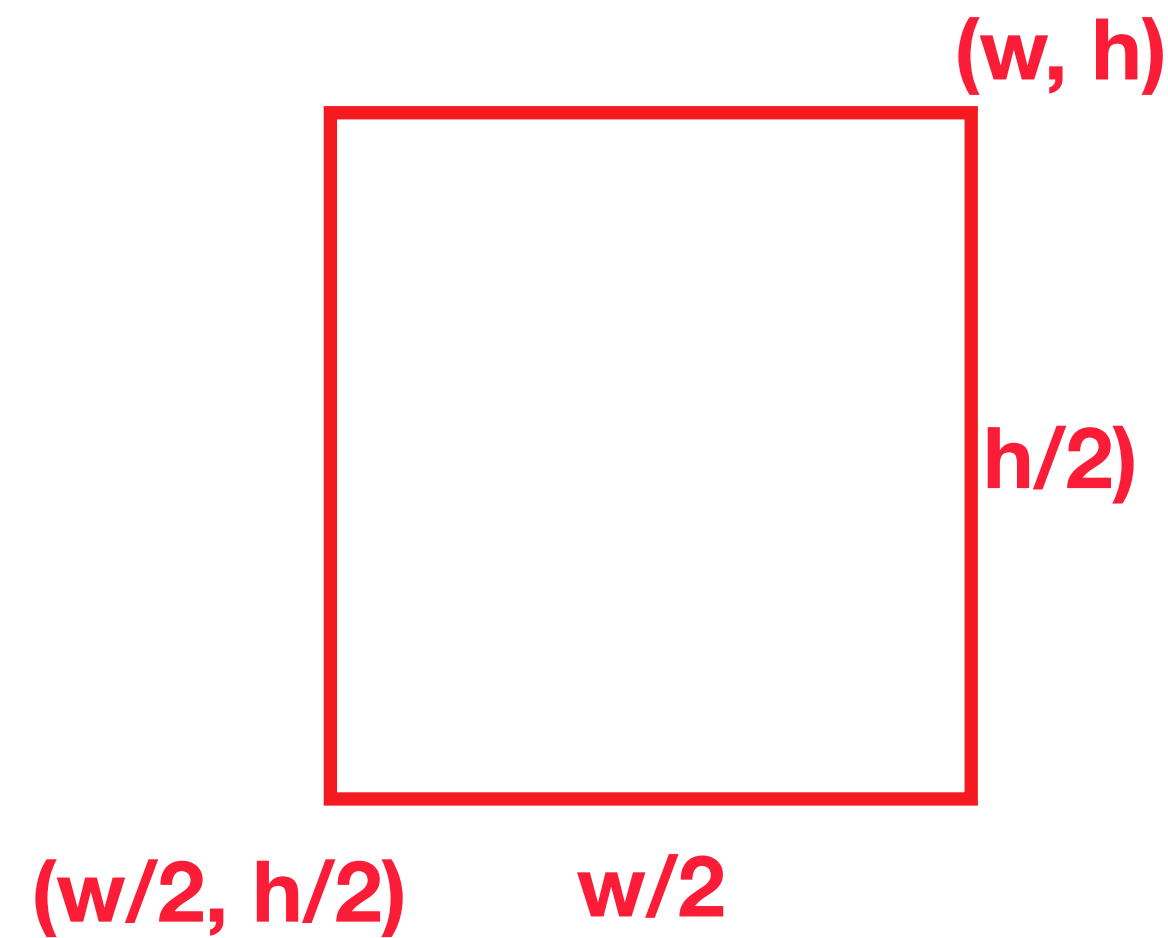
ViewportOrtho.c

Ortho(0, w, 0, h) - Black window
(Actual screen)



(-250, 250), (-250, -250),
(250, -250), (250, 250)

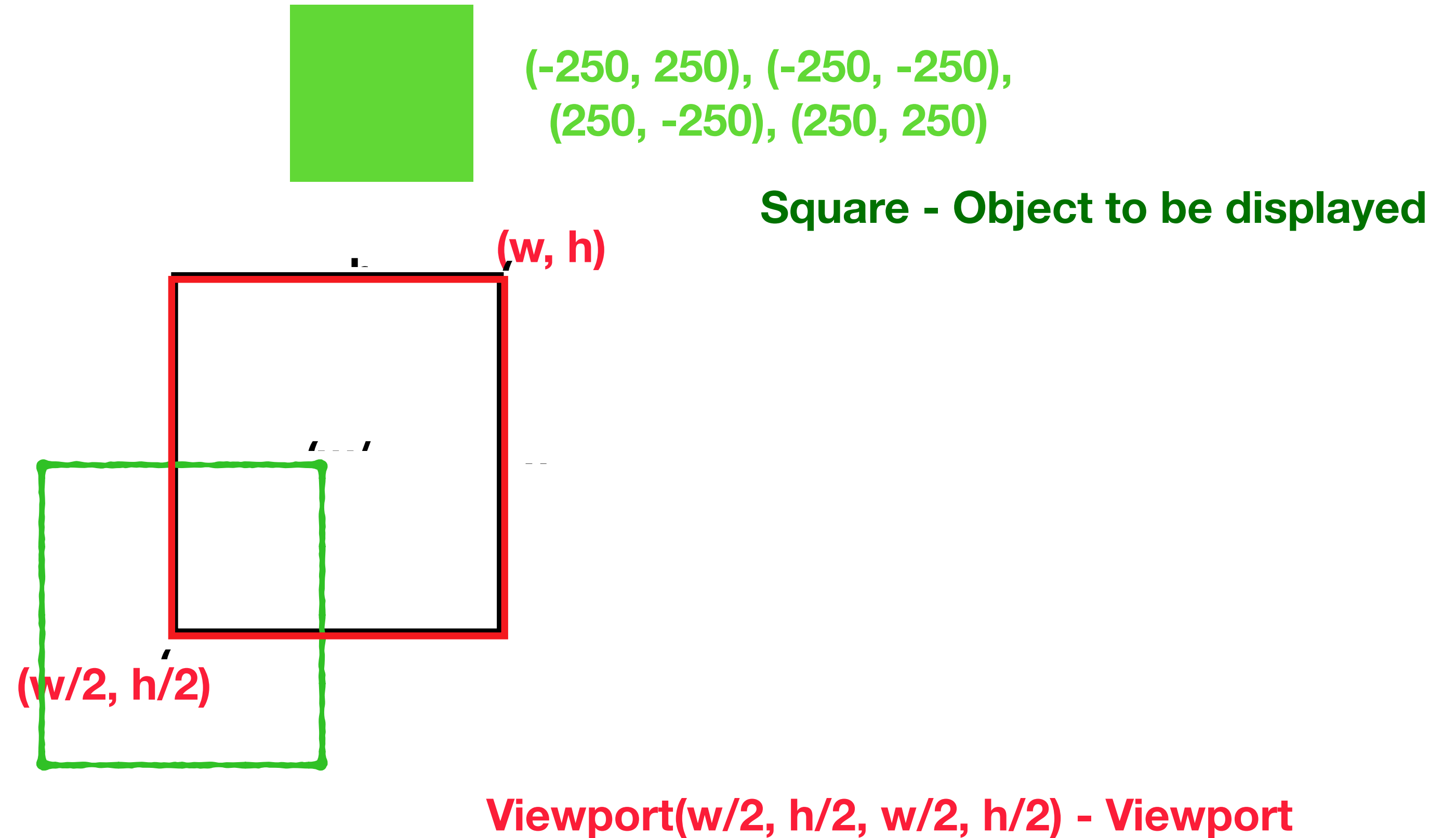
Square - Object to be displayed



Viewport(w/2, h/2, w/2, h/2) - Viewport

ViewportOrtho.c

Ortho(0, w, 0, h) - Black window
(Actual screen)



Change coordinates in display function

To experiment with `glViewport()` and `glOrtho()`

```
void
display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glColor3f(0.0, 0.0, 1.0);  /* blue */
    glBegin(GL_POLYGON);
    glVertex2i(-250, 250);
    glVertex2i(250, 250);
    glVertex2i(250, -250);
    glVertex2i(-250, -250);
    glEnd();

    glFlush();  /* Single buffered, so needs a flush. */
}
```


glOrtho() and glViewport()

window and screen/viewport coordinates

```
void
reshape(int w, int h)
{
    /* Because Gil specified "screen coordinates" (presumably with an
       upper-left origin), this short bit of code sets up the coordinate
       system to correspond to actual window coordinates. This code
       wouldn't be required if you chose a (more typical in 3D) abstract
       coordinate system. */

    glViewport(0, 0, w, h);          /* Establish viewing area to cover entire window. */
    glMatrixMode(GL_PROJECTION);    /* Start modifying the projection matrix. */
    glLoadIdentity();               /* Reset project matrix. */
    glOrtho(0, w, 0, h, -1, 1);     /* Map abstract coords directly to window coords. */
}
```

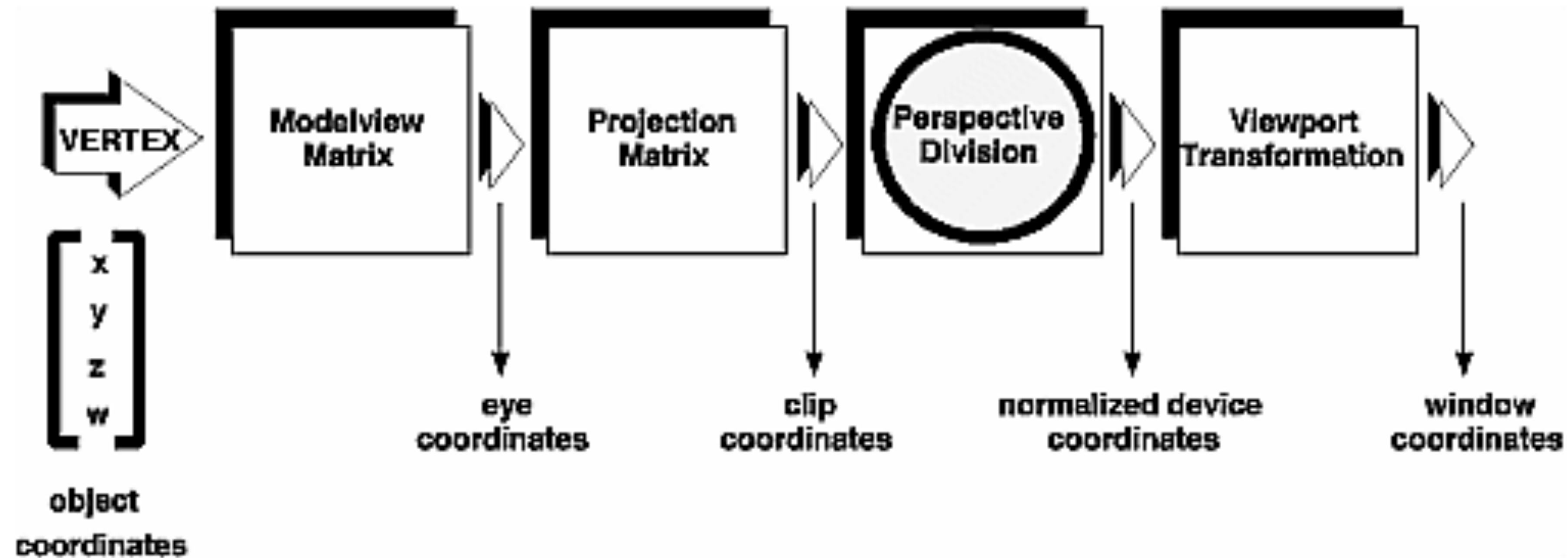
Transformations

- Translation
- Rotation
- Scaling

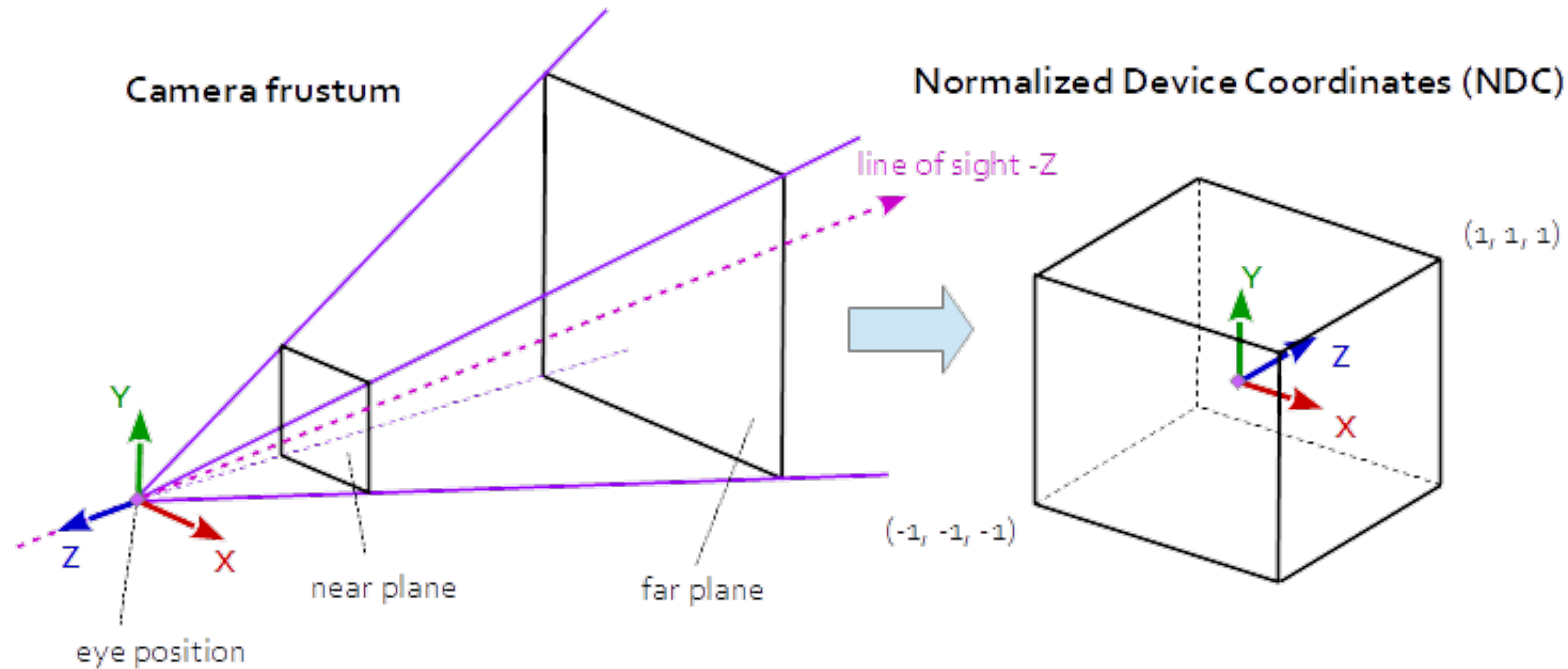
Programming Transformations

- Prior to rendering, view, locate, and orient:
 - eye/camera position
 - 3D geometry
- Manage the matrices
- Combine transformations
- Transformation matrices must be defined prior to any vertices.
- OpenGL provides matrix stacks for each type of supported matrix (ModelView, projection etc.) to store matrices.

Transformation Pipeline (From redbook)



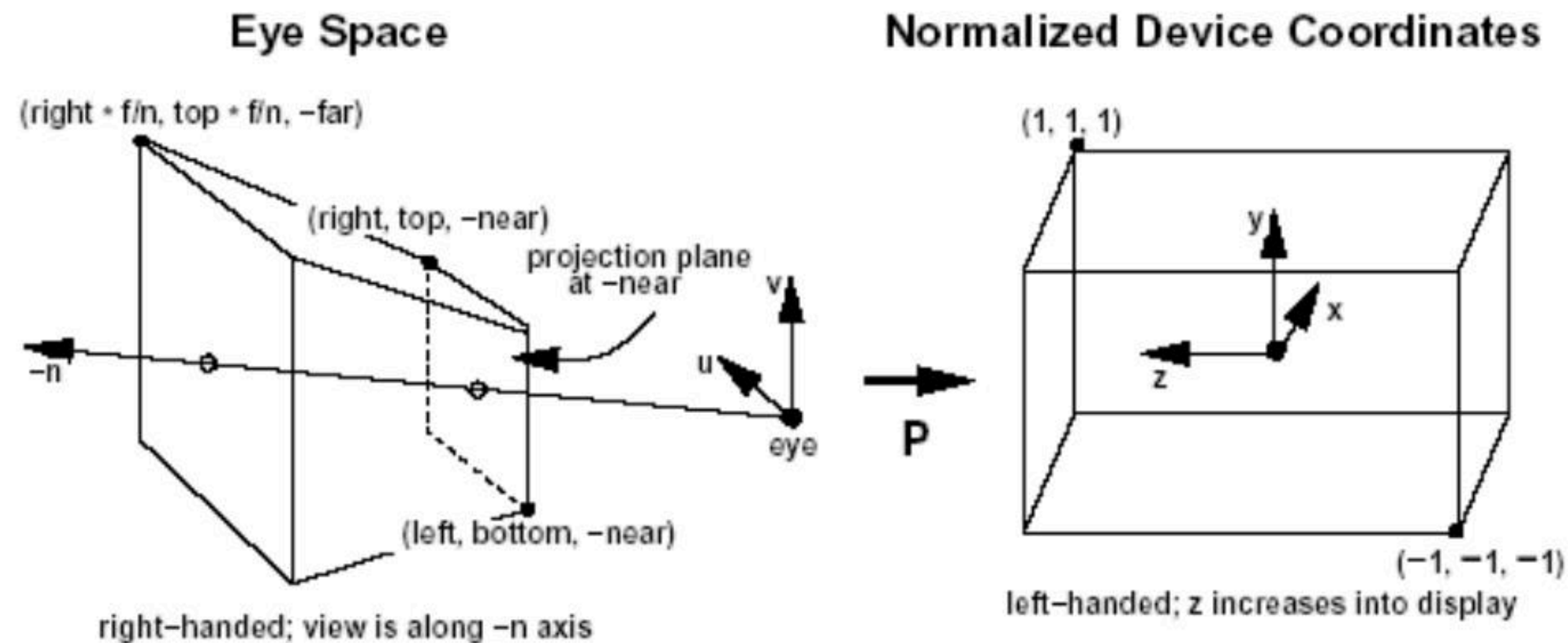
Normalized device coordinates



NCD

Normalized Device Coordinates

- Clipping is more efficient in a rectangular, axis-aligned volume: $(-1,-1,-1) \rightarrow (1,1,1)$ *OR* $(0,0,0) \rightarrow (1,1,1)$



gluPerspective()

*void gluPerspective(GLdouble fovy, GLdouble aspect,
GLdouble near, GLdouble far);*

