Drawing a Rectangle

Lecture 4
Red Book: OpenGL Initialization and Rendering

Robb T. Koether

Hampden-Sydney College

Wed, Sep 2, 2015

Outline

Outline

Drawing a Rectangle

- In earlier versions of OpenGL, drawing a rectangle was quite simple.
  - Announce that you were going to draw a rectangle:

    ```
    glBegin(GL_RECT);
    ```

  - Pass the vertices one by one:

    ```
    glVertex2f(0.0, 1.0)
    ```

    Etc.
- It is a bit more complicated now.

Drawing a Rectangle

- The three basic steps are
  - Create an array of vertex attributes.
  - Create a vertex buffer object.
  - Create a vertex array object (stored in the buffer).
  - Issue the draw command.

Outline

Vertex Attributes

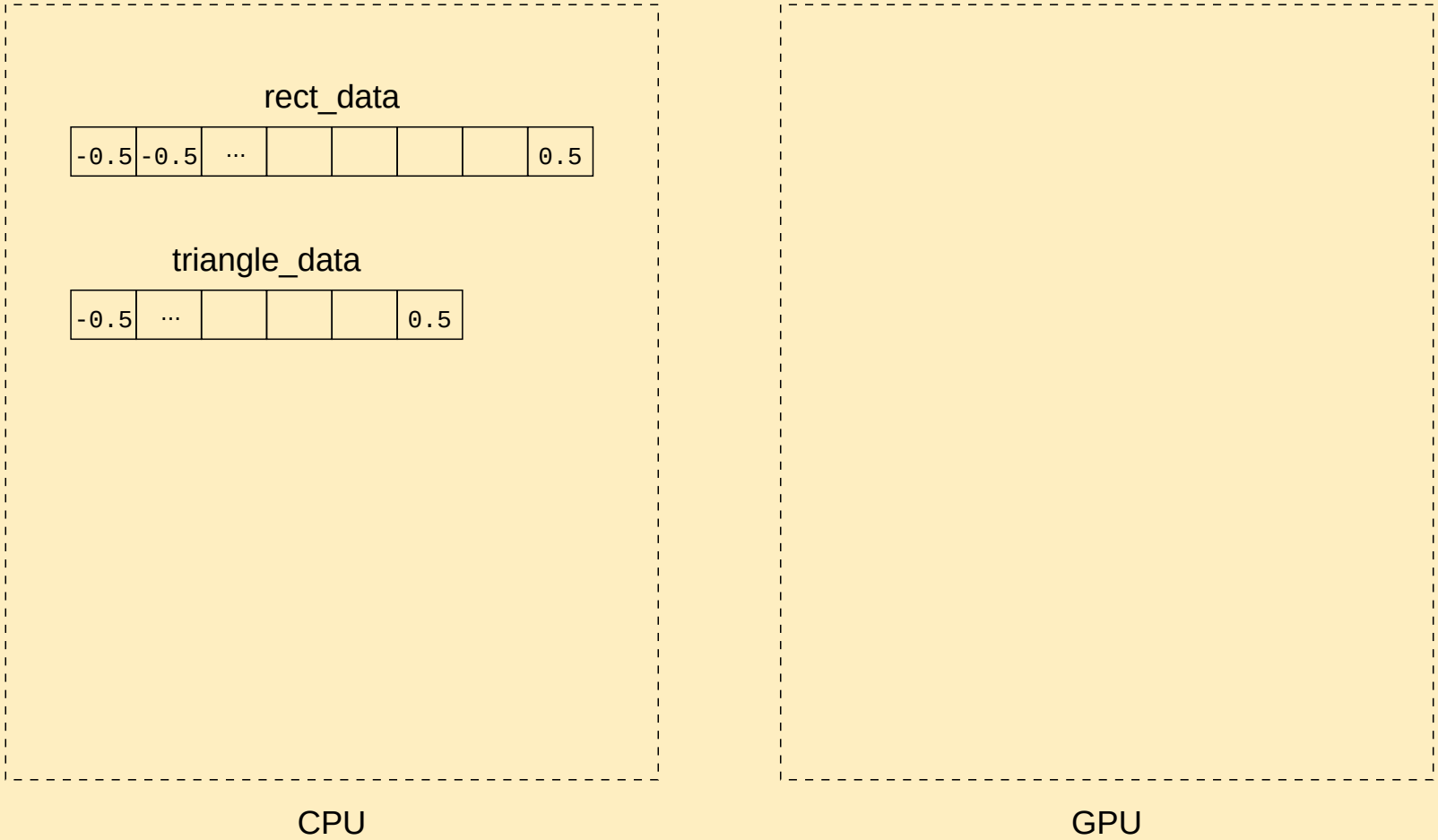
Vertex Attributes

```
GLfloat rect_data[] =
{
    -0.5f, -0.5f,
     0.5f, -0.5f,
     0.5f,  0.5f,
    -0.5f,  0.5f
};
GLfloat triangle_data[] = {...};
```

- In this first example, the only vertex attributes will be the coordinates of the 2D vertices.

Vertex Buffer Objects

Vertex Buffer Object



Outline

Vertex Buffer Objects

- A vertex buffer object (VBO) is a data structure in the GPU that contains data related to the vertices of an object.
  - Coordinates of the vertices.
  - Their color.
  - Normal vectors.
  - Etc.

Vertex Buffer Objects

- To use a VBO, we must do three things.
  - Generate an ID number for the buffer object.
  - “Bind” the buffer object, i.e., associate the ID number with the buffer object and make it the current (or active) buffer.
  - Copy the vertex data to the buffer object.

Vertex Buffer Objects

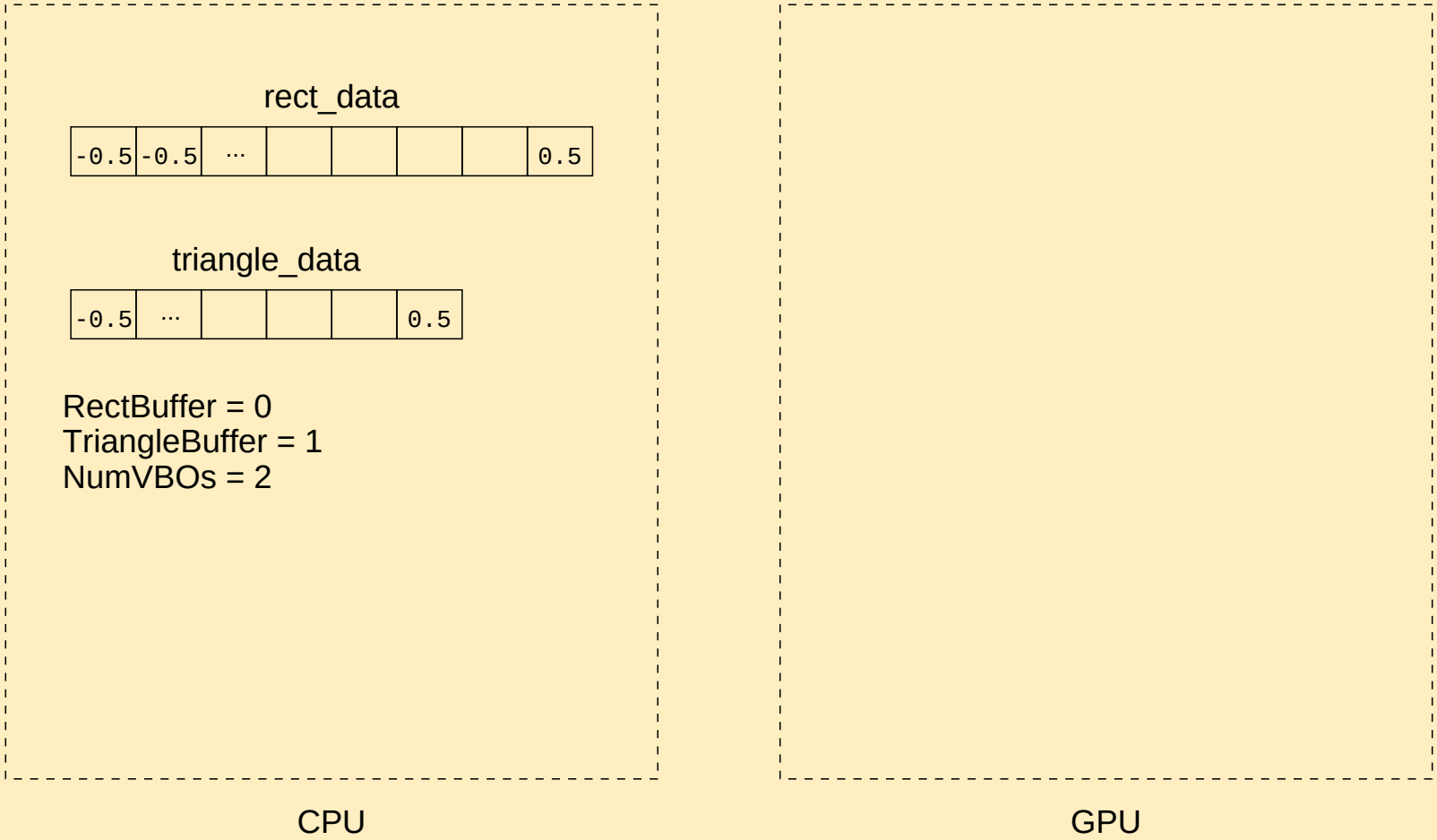
Symbolic Names for the VBOs

```
enum {RectBuffer, TriangleBuffer, NumVBOs};
```

- The **enum** statement will assign the values `0`, `1`, and `2` to `RectBuffer`, `TriangleBuffer`, and `NumVBOs`, respectively.

Vertex Buffer Objects

Vertex Buffer Object



Vertex Buffer Objects

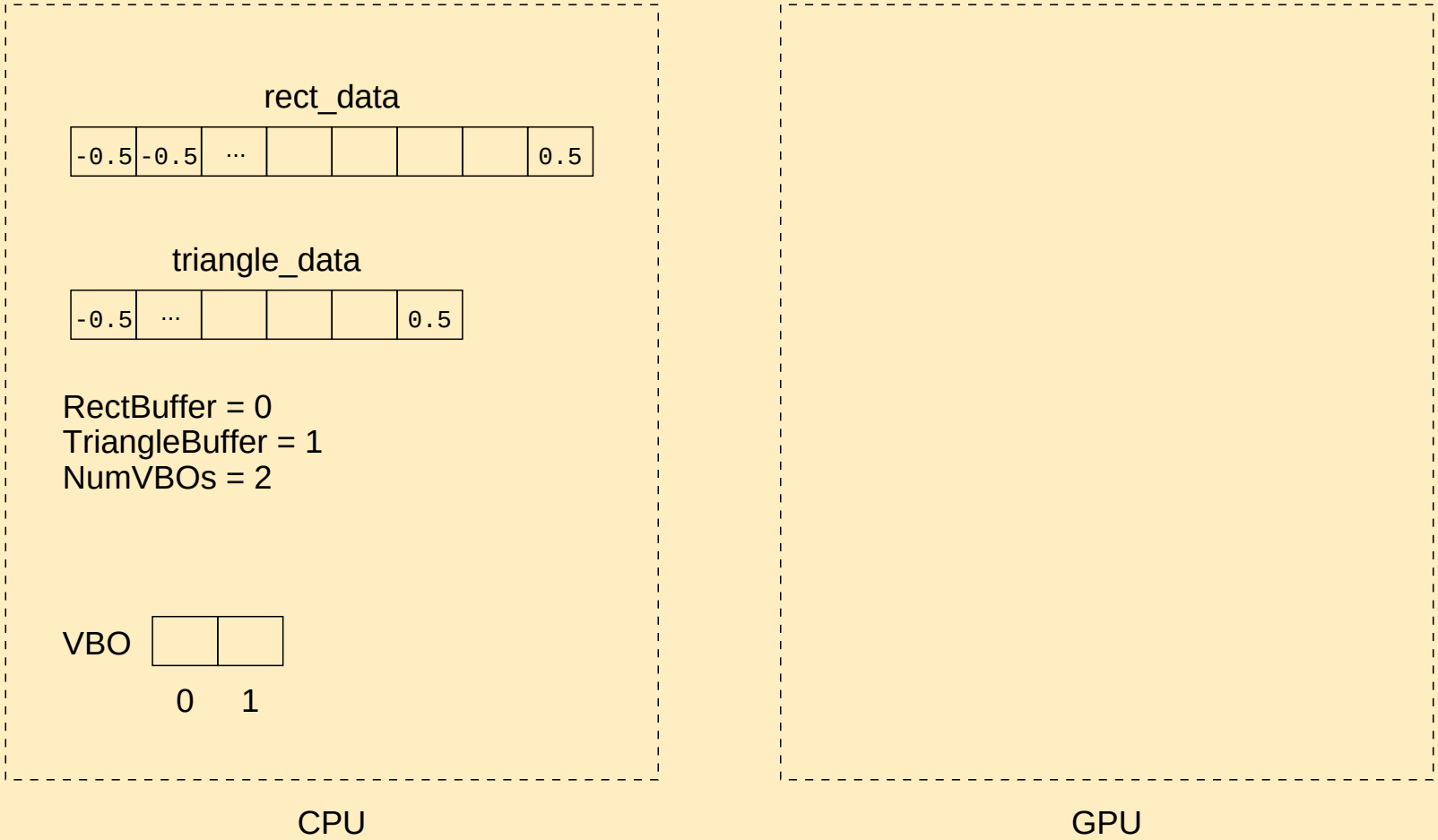
Array of VBO IDs

```
GLuint VBO[NumVBOs];
```

- The array `VBO` will store the ID numbers (to be assigned by OpenGL) of the buffer objects.
- The **`enums`** `RectBuffer` and `TriangleBuffer` are symbolic names for the indexes of the IDs in the array `VBO`.

Vertex Buffer Objects

Vertex Buffer Object



Vertex Buffer Objects

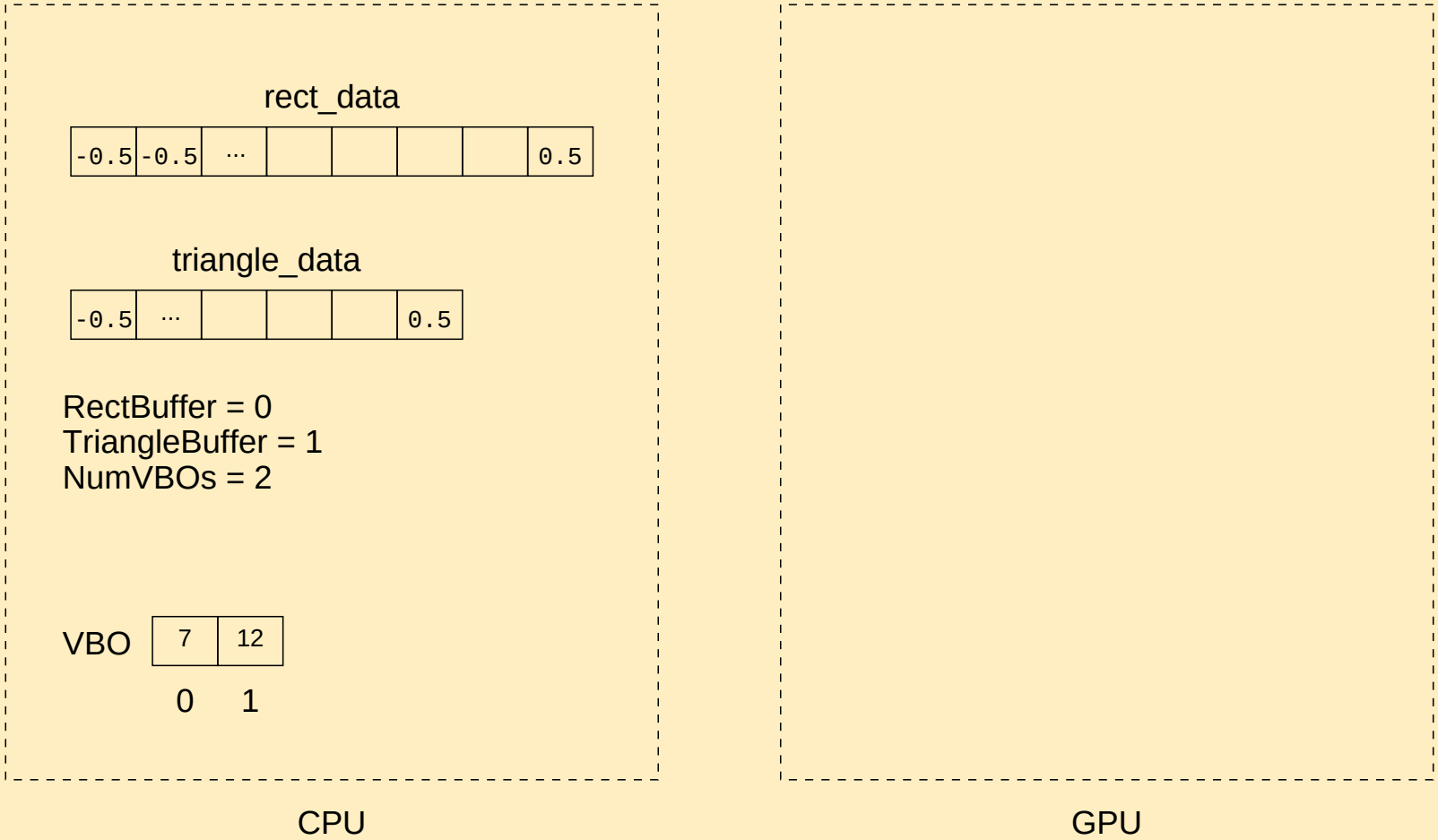
Vertex Buffer Object

```
glGenBuffers(NumVBOs, VBO);
```

- Generate ID numbers for each of the buffers and store them in `VBO[0]` and `VBO[1]`, also known as `VBO[RectBuffer]` and `VBO[TriangleBuffer]`.

Vertex Buffer Objects

Vertex Buffer Object



Vertex Buffer Objects

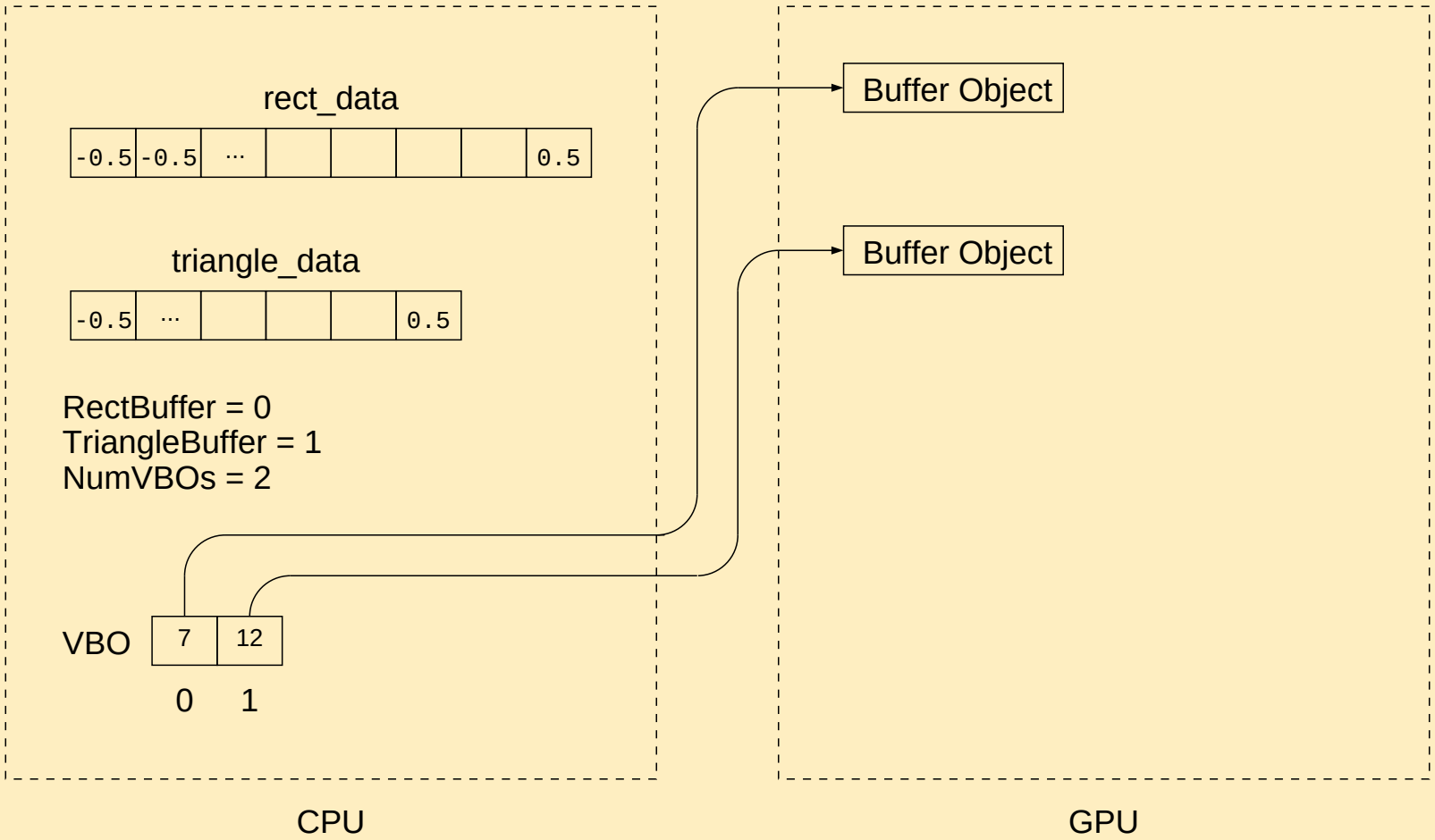
Vertex Buffer Object

```
glBindBuffer(GL_ARRAY_BUFFER, VBO[RectBuffer]);
```

- `glBindBuffer()` binds (associates) the buffer ID `VBO[RectBuffer]` to a new buffer object in the GPU and makes that buffer object the current buffer.
- When `glBindBuffer()` is called subsequently with the same buffer ID, it simply makes that buffer object the current one.

Vertex Buffer Objects

Vertex Buffer Object



Vertex Buffer Objects

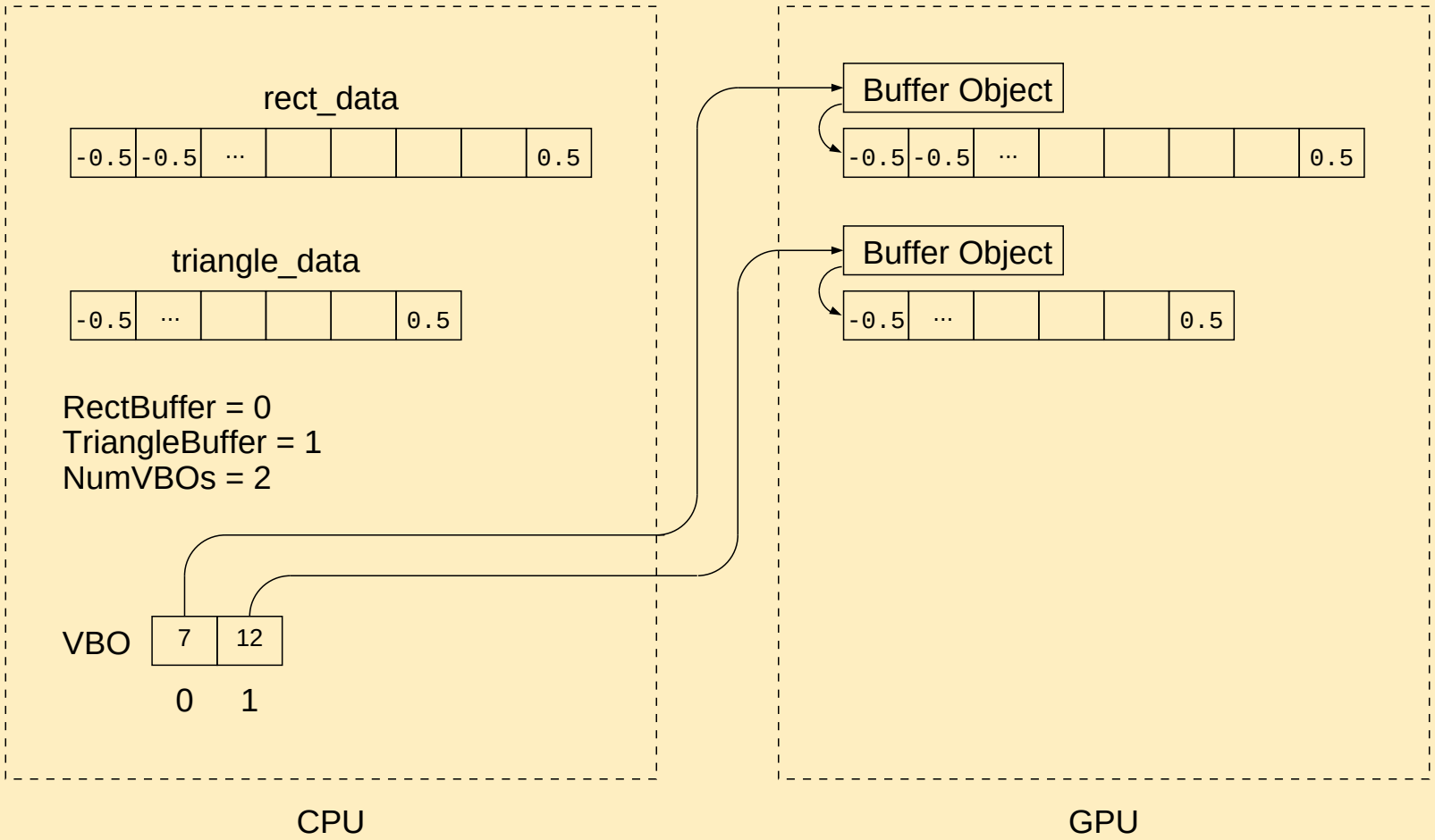
Vertex Buffer Object

```
glBufferData(GL_ARRAY_BUFFER, sizeof(rect_data),
    rect_data, GL_STATIC_DRAW);
```

- `glBufferData()` copies the data from `rect_data` into the *current* buffer.
- The parameter `GL_STATIC_DRAW` informs OpenGL that this data will change infrequently, if at all, allowing for greater efficiency.
- If an object will be modified frequently, then we should use `GL_DYNAMIC_DRAW`.

Vertex Buffer Objects

Vertex Buffer Object



Outline

Vertex Array Objects

- A vertex array object (VAO) describes the structure imposed on the data stored in the buffer object.
- We follow a similar pattern with VAOs as we did with VBOs.
- To use a VAO, we must do three things.
  - Generate an ID number for the vertex array object.
  - “Bind” that vertex array object to the active buffer object.
  - Describe the structure (i.e., attributes) of the data in the buffer.
  - Enable the vertex attributes.
- Then we are ready to draw the object.

Vertex Array Objects

Symbolic Names for the VAOs

```
enum {Rect, Triangle, NumVAOs};
enum {vPosition = 0};
```

- We use an enumerated type to create symbolic names for the VAOs.
- We also use an enumerated type to create symbolic names for the vertex attributes.
- In this example, the only attribute is the position.

Vertex Array Objects

Array of VAO IDs

```
GLuint VAO[NumVAOs];
```

- Create an array of vertex array objects.
- As with the VBOs, this array will hold the ID number of the VAOs in the GPU.

Vertex Array Objects

Vertex Array Object

```
glBindVertexArray(VAO[Rect]);
```

- `glBindVertexArray()` will create vertex array objects in the GPU and store their IDs in the `VAO` array.
- This statement will store the ID for the rectangle VBO in `VAO[0]`.
- It is necessary that `VBO[RectBuffer]` be the current VBO.

Vertex Array Objects

Vertex Array Object

```
glVertexAttribPointer(vPosition, 2, GL_FLOAT,
    GL_FALSE, 0, BUFFER_OFFSET(0));
```

- This statement associates the attribute ID `vPosition` (i.e., `0`) with the following information.
  - The `2` indicates the number of objects that constitute a single attribute (2 floats = a 2D point).
  - `GL_FLOAT` tells the type of object in the attribute.
  - `GL_FALSE` tells the GPU not to "normalize" the data (more on that later).
  - the `0` is the stride, i.e., the number of bytes to skip over from one attribute value to the next. The value `0` means that the data are packed.
  - `BUFFER_OFFSET(0)` gives the offset, in bytes, to the first attribute value.

Vertex Array Objects

Enable the Attribute

```
glEnableVertexAttribArray(vPosition);
```

- This statement makes the attribute with index `vPosition` (i.e., `0`) active.
- The values will be sent to the shader programs.

Vertex Array Objects

Vertex Array Object

```
glDrawArrays(GL_TRIANGLE_FAN, 0, 4);
```

- Invoke the `glDrawArrays()` function, with parameters
  - The type of object to draw (e.g., `GL_TRIANGLE_FAN`).
  - The starting index in the array.
  - The number of vertices.
- This example will draw a rectangle.

Vertex Array Objects

- The possible types of object to draw are
  - `GL_POINTS` – individual points
  - `GL_LINES` – line segments
  - `GL_TRIANGLES` – triangles
  - `GL_LINE_STRIP` – line segments joined
  - `GL_LINE_LOOP` – line segments joined in a circuit
  - `GL_TRIANGLE_FAN` – triangles fanning out from a point
  - `GL_TRIANGLE_STRIP` – triangles forming a strip

Outline

Assignment 3

Assignment 3

- Run the `DrawRectangle.cpp` program.
- Change `GL_TRIANGLE_FAN` to `GL_LINE_LOOP` and run it again.
- Add two vertices to `rect_data` and draw all 6 vertices as a line loop.
- Change the coordinates in `rect_data` so that the program draws a plus sign (with 12 corners) as a line loop.
- Draw the plus sign as a filled figure (triangle fan).

Outline

Assignment

Assignment

- Read pp. 16 - 28 in The Red Book.