#3: Hierarchical Transforms. Geometric Calculations

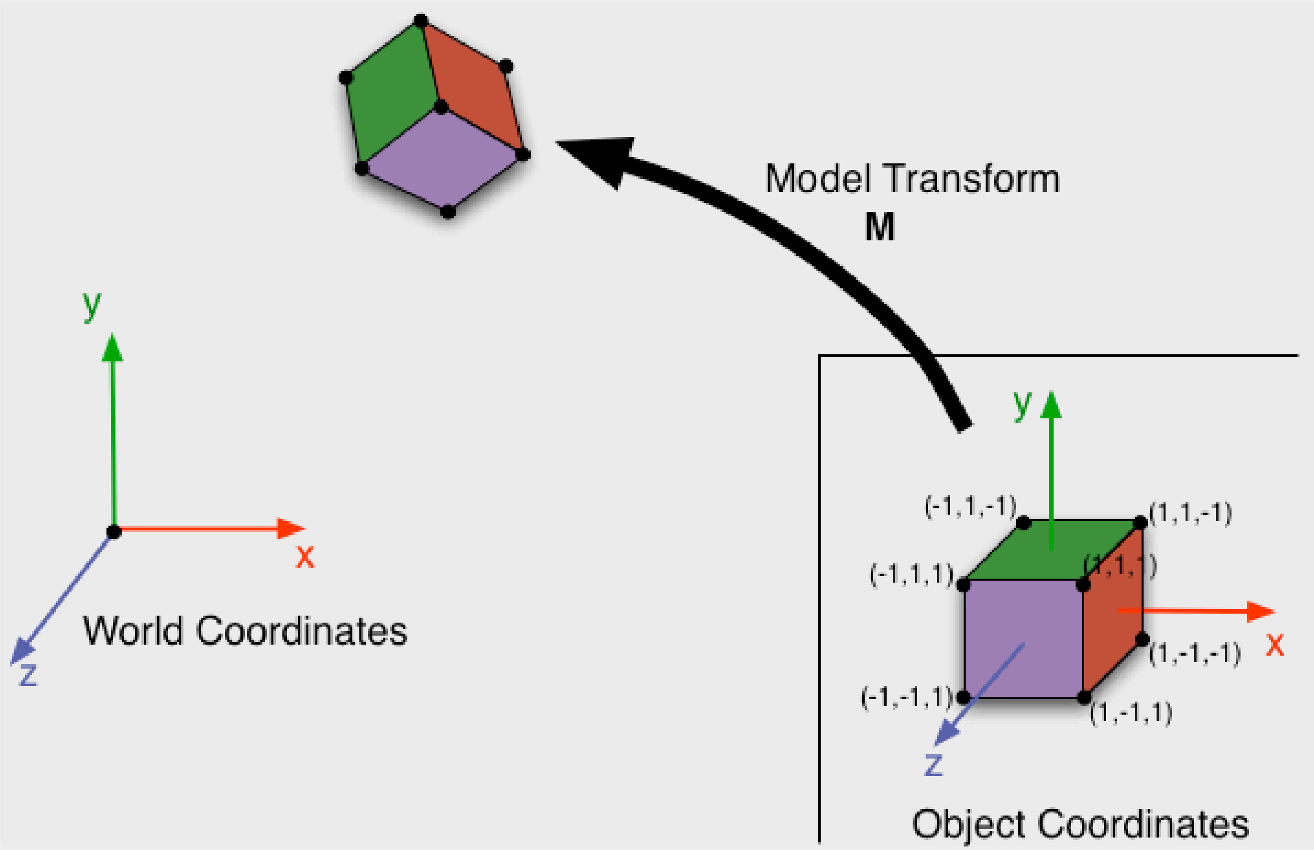
CSE167: Computer Graphics

Instructor: Ronen Barzel

UCSD, Winter 2006

Object and World Coordinates

- In project1, constructed matrix to transform points of cube
  - Cube defined using (-1,1,1), …
  - Transformed each point to final position



Object Coordinates

- Each object is defined using some convenient coordinates
  - Often "Axis-aligned". (when there are natural axes for the object)
  - Origin of coordinates is often in the middle of the object
  - Origin of coordinates is often at the "base" or corner of the object
  - E.g. cube in project1 was -1,-1,1… could also use just 0,1 Axes could be lined up any way.
    - Model of a book: put the Z axis along the spine? Front-to-back?
- No "right" answer. Just what's most convenient for whomever builds model
- Notice: build, manipulate object in object coordinates
  - Don't know (or care) where the object will end up in the scene.
- Also called
  - *Object space*
  - *Local coordinates*

World Coordinates

- The common coordinate system for the scene
- Also called World Space
- Also chosen for convenience, no “right” answer.
  - Typically if there’s a ground plane, it’s XY horizontal and Z up
    - That’s most common for people thinking of models
    - I tend to use it a lot

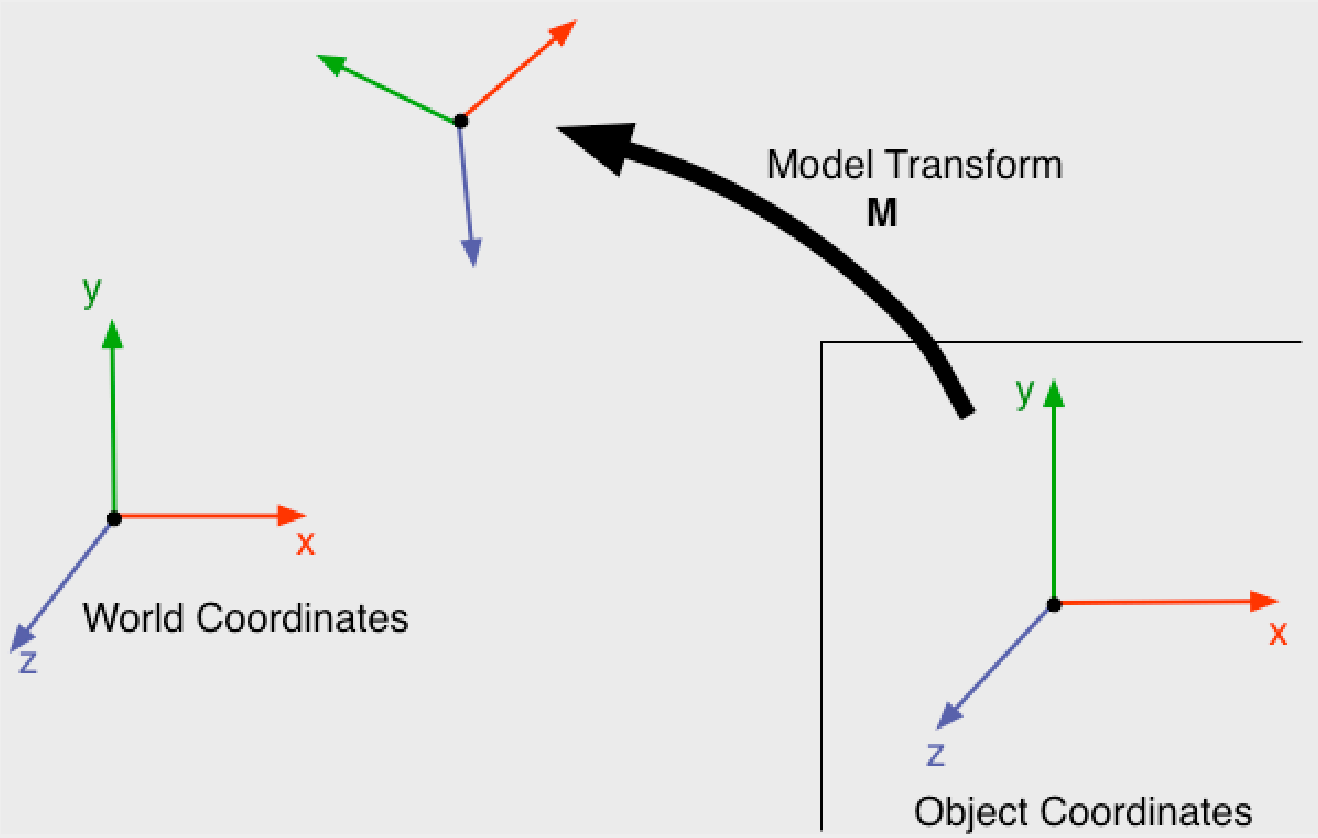
- Aside: *Screen Coordinates*
  - X to the right, Y up, Z towards you
    - That’s the convention when considering the screen (rendering)
    - Handy when drawing on the blackboard, slides
    - In project1, World Coordinates == Screen Coordinates

Placing object in the world

- Bundle together a single composite transform.
- Known as:
  - *Model Matrix*
  - *Model Transform*
  - *World Matrix*
  - *World Transform*
  - *Model-to-World Transform*
  - *Local-to-World Matrix*
  - In OpenGL: included in MODELVIEW matrix (composed model & view matrix)
- Transforms each point

Placing object coordinates in the world

- Place the coordinate frame for the object in the world
  - Don’t know or care about the shape of the object
  - World matrix columns = object’s frame in world coordinates

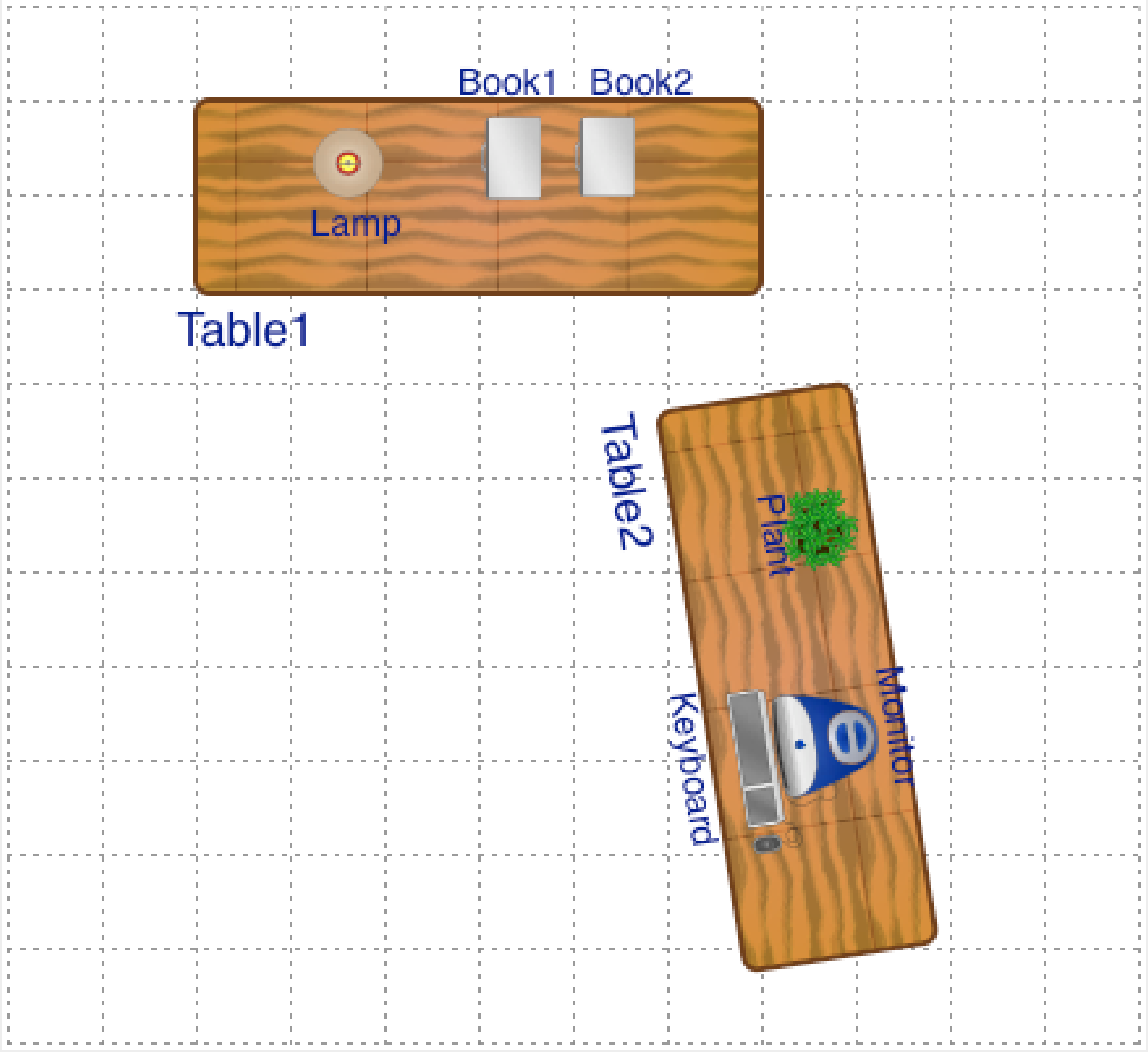


Relative Transformations

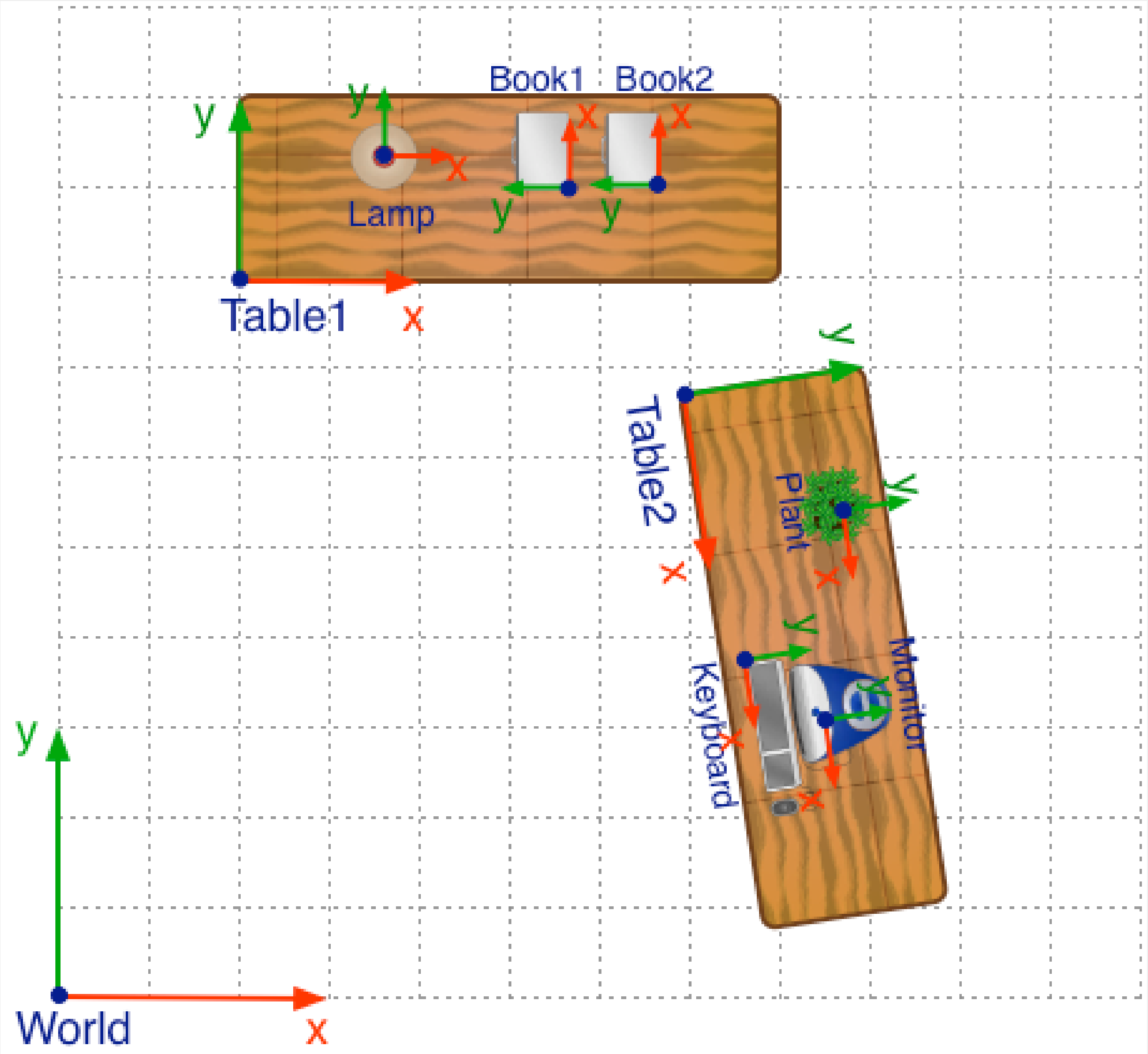
- Until now, used a separate world matrix to place each object into the world separately.
- But usually, objects are organized or grouped together in some way
- For example…
  - A bunch of moons and planets orbiting around in a solar system
  - Several objects sitting on a tray that is being carried around
  - A hotel with 1000 rooms, each room containing a bed, chairs, table, etc.
  - A robot with torso and jointed arms & legs
- Placement of objects is described more easily relative to each other rather than always in world space

Sample Scene

Schematic Diagram (Top View)



Top view with Coordinates



Relative Transformations

- Put the objects on the tables
  - Each table has a simple coordinate system
  - E.g. Book1 at (3.75,1,0) on Table1's top
  - E.g. Keyboard at (3,.5,0) on Table2's top
  - Don't care where the tables are in order to do this part
- Put the tables in the room
  - Books etc. should end up in the right place
- How do we do this…?

Current coordinate system

- In our code, we maintain a “current coordinate system”
- Everything we draw will be in those coordinates
- I.e. we keep a variable with a matrix known as the “current transformation matrix” (CTM)
  - Everything we draw we will transform using that matrix
  - Transforms from current coordinates to world coordinates

Drawing with a CTM

- Old drawCube:

```
drawCube(Matrix M) {
   p1 = M*Point3( 1,-1, 1);
   p2 = M*Point3( 1,-1,-1);
   p3 = M*Point3( 1, 1,-1);
   p4 = M*Point3( 1, 1, 1);
   p5 = M*Point3(-1,-1, 1);
   p6 = M*Point3(-1,-1,-1);
   p7 = M*Point3(-1, 1,-1);
   p8 = M*Point3(-1, 1, 1);
        .
        .
        .
    }
```

- New drawCube:

```
// global CTM
drawCube() {
   p1 = CTM*Point3( 1,-1, 1);
   p2 = CTM*Point3( 1,-1,-1);
   p3 = CTM*Point3( 1, 1,-1);
   p4 = CTM*Point3( 1, 1, 1);
   p5 = CTM*Point3(-1,-1, 1);
   p6 = CTM*Point3(-1,-1,-1);
   p7 = CTM*Point3(-1, 1,-1);
   p8 = CTM*Point3(-1, 1, 1);
        .
        .
        .
    }
```

Using a CTM

- As we go through the program, we incrementally update the CTM
- Start with the current coordinates=world coordinates
  - CTM = I
- Before we draw an object, we update the CTM
  - from the current location to the object's location
  - We perform a relative transformation.
  - The CTM accumulates the full current-to-world transformation.
- Draw from the outside in.
  - Draw containers before the things they contain.

Top view, just frames

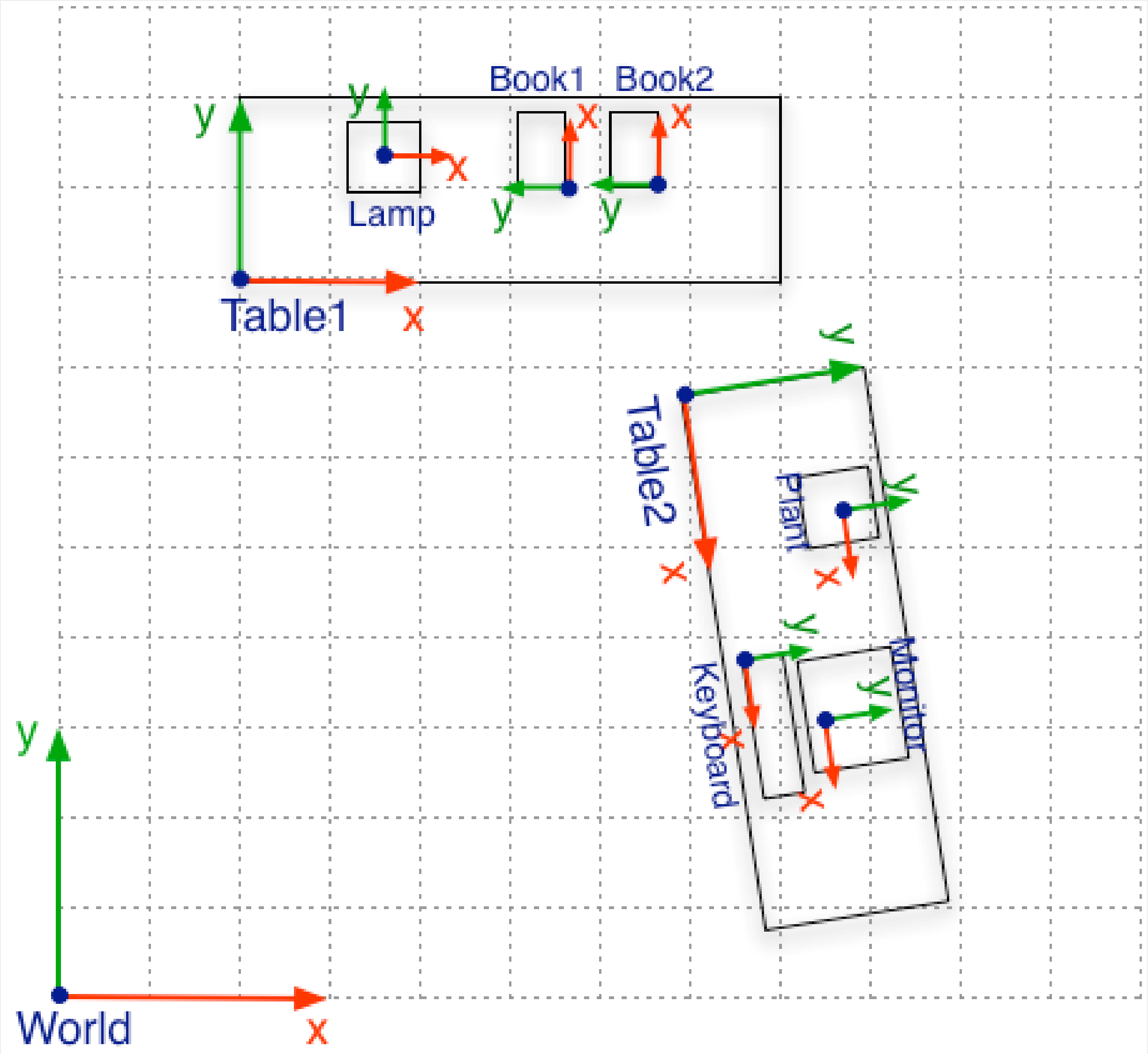
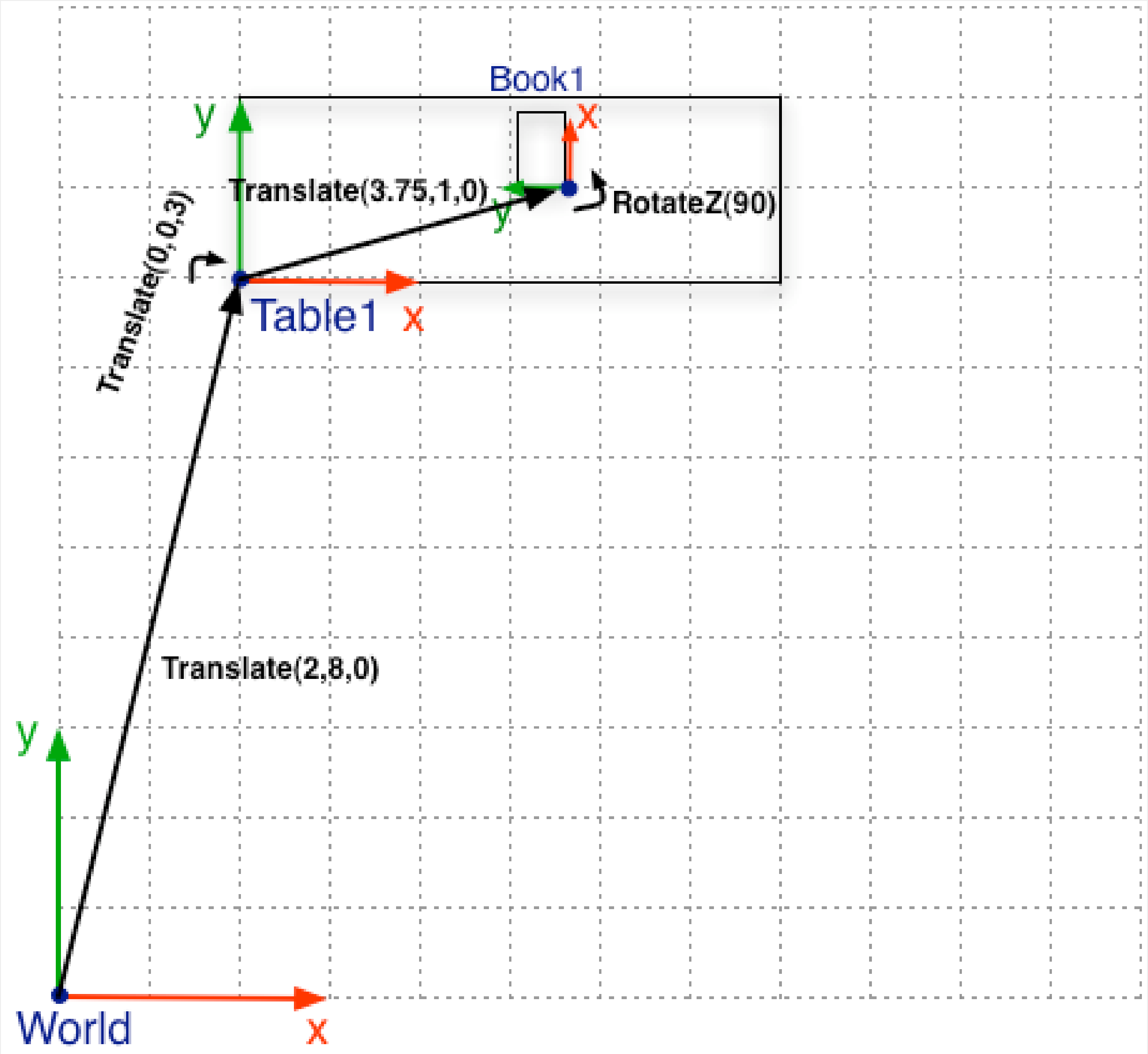


Table1 and Book1



Draw Table1 and Book1

```
// Start in World coords, on floor of room
CTM = Matrix::IDENTITY;

// Move to Table1 position, draw table
CTM = CTM*Matrix.MakeTranslate(2,8,0);
drawTable();

// Move up to tabletop height
CTM = CTM*Matrix.MakeTranslate(0,0,3);

// Move to Book1 position & orientation, draw
CTM = CTM*Matrix.MakeTranslate(3.75,1,0);
CTM = CTM*Matrix.MakeRotateZ(90);
drawBook();
```

Simplify the idiom

- Routines that affect the CTM:
  - LoadIdentity () { CTM = Matrix::IDENTITY }
  - Translate(V) { CTM = CTM*Matrix::MakeTranslate(V) }
  - RotateZ(angle) { CTM = CTM*Matrix::MakeRotateZ(angle) }
  - Etc…
  - Transform(M) { CTM = CTM*M }

Draw Table1 and Book, redux

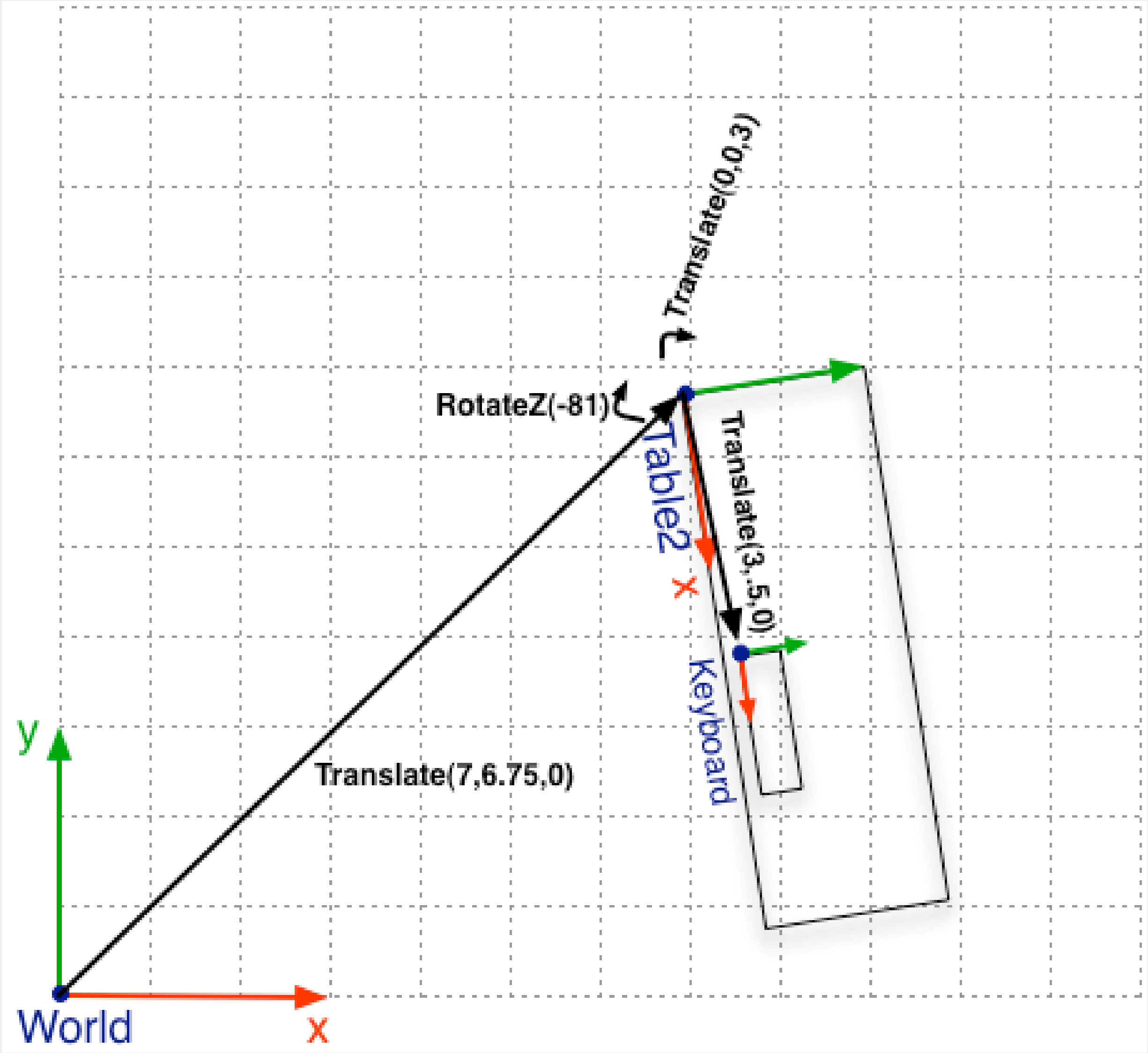
```
// Start in World coords, on floor of room
LoadIdentity();

// Move to Table1 position, draw table
Translate(2,8,0);
drawTable();

// Move up to tabletop height
Translate(0,0,3);

// Move to Book1 position & orientation, draw
Translate(3.75,1,0);
RotateZ(90);
drawBook();
```

Table2 and Keyboard



Draw Table2 and Keyboard

```
// Start in World coords, on floor of room
LoadIdentity();

// Move to Table2 position & orientation, draw
Translate(2,8,0);
RotateZ(-81);
drawTable();

// Move up to tabletop height
Translate(0,0,3);

// Move to Keyboard position, draw
Translate(3,0.5,0);
drawKeyboard();
```

What about drawing entire scene?

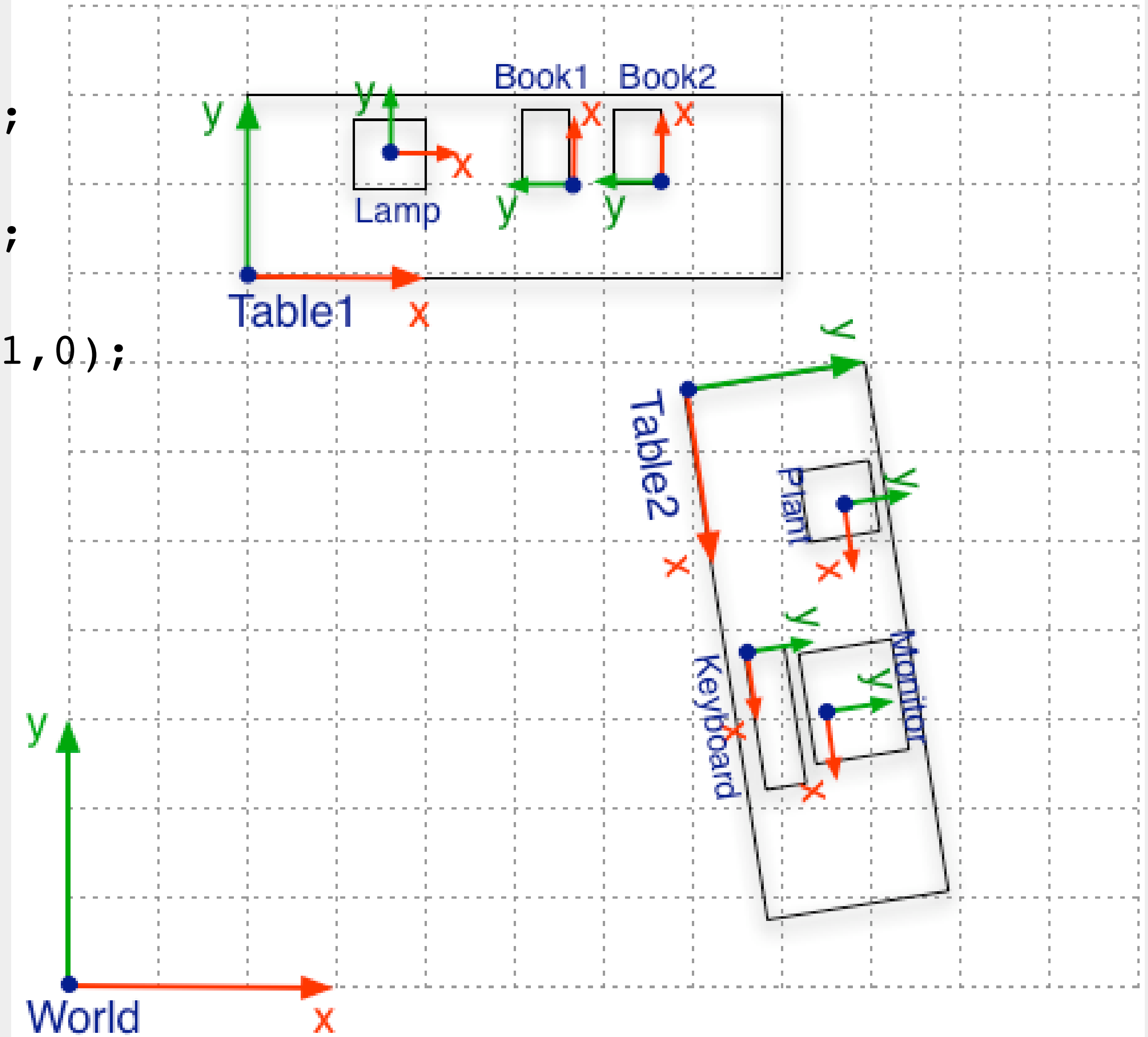
- After we drew Book1 or Keyboard, our coordinate system had moved deep into the world somewhere.
- How do we get back…?
  - To the tabletop coordinates so we can place another book?
  - To the room coordinates so we can place another table?
- Don’t want to start over at the beginning for each object.
- At each stage, need to remember where we are so we can get back there

Keep a Stack for the CTM

- Add two more routines:
  - PushCTM() -- saves a copy of the CTM on the stack
  - PopCTM() -- restores the CTM from the stack

Draw whole scene, hierarchically

```
PushCTM();
 Translate(2,8,0);
 drawTable()
 Translate(0,0,3);
 PushCTM();
  Translate(3.75,1,0);
  RotateZ(90);
  drawBook()
 PopCTM();
 PushCTM();
  Translate(…);
  Rotate(…);
  drawBook();
 PopCTM();
 …etc…
PopCTM()
…etc…
```



Hierachical grouping within a model

- Model can be composed of parts
  - Draw parts using Push & Pop CTM

```
drawTable(){
    PushCTM() // save
        PushCTM() // draw leg1
          Translate(…);
          drawLeg();
        PopCTM();
        PushCTM() // draw leg2
          Translate(…);
          drawLeg();
        PopCTM();
        …etc leg3 & leg4…
        PushCTM(); // draw top
          Translate(…);
          drawTableTop();
        PopCTM();
    PopCTM() // restore
  }
```

  - Has no effect outside this routine.

Access something in the middle?

- CTM always contains the complete Local-to-world transform for what we're currently drawing.
- Sometimes need to hold on to copy of CTM in the middle

```
pushCTM();
   …stuff…
  pushCTM();
    …transform…
    Book1Matrix = CTM;
    drawBook();
  popCTM();
  …stuff…
popCTM();
```

- Later in code, mosquito lands on Book1

```
pushCTM();
  LoadMatrix(Book1Matrix);
  Translate(…);
  drawMosquito();
popCTM();
```

CTM and matrix stack in OpenGL

- OpenGL provides
  - glTranslatef(…)
  - glRotatef(…)
  - glPushMatrix()
  - glPopMatrix()
- (But don't use them for proj2--need to know how to do it yourself)
- Actually, other properties, such as color, are also part of "current state" and can be pushed and popped.

Thinking top-down vs bottom-up

- Transforms for World-to-Keyboard (ignoring pushes, pops, etc.):
  1. Translate(2,8,0)
  2. RotateZ(-81)
  3. Translate(0,0,3)
  4. Translate(3,0.5,0)
  5. drawKeyboard()
- Top-down: transform the coordinate frame:
  - Translate the frame, then rotate it, then translate twice more, then draw the object
- Bottom-up: transform the object:
  - Create a keyboard, translate it in X&Y, then in Z, then rotate about the origin, then translate again
- Both ways give same result
- Both ways useful for thinking about it.

Another example:

- Sample sequence:
  1. RotateZ(45)
  2. Translate(0,5,0)
  3. Scale(2,1,1)
  4. drawCube()
- Top-down: transform a coordinate frame:
  - rotate it 45 degrees about its origin, then translate it along its Y, then stretch it in X, then draw the primitive.
- Bottom-up: transform the object
  - create a square, then scale it in X then translate it along the Y axis, then rotate 45 degrees about the origin.
- Both ways useful