Scripting Languages

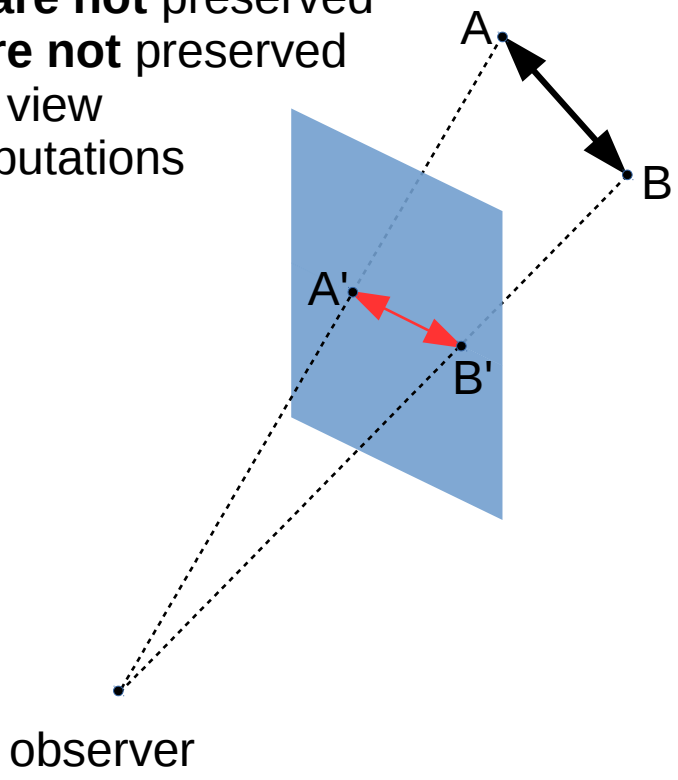
Simple approach to 3D graphics:

- basic 3D projection
- drawing 3D objects
- rotating 3D objects
- exercises

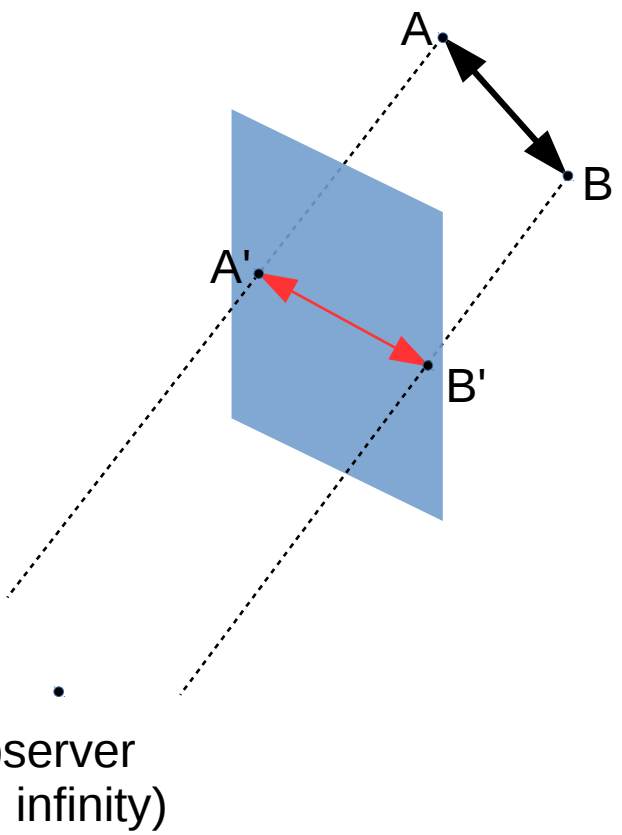
Viewing 3D graphics

- conversion: (x, y, z) → (x', y')
- projection onto 2D plane

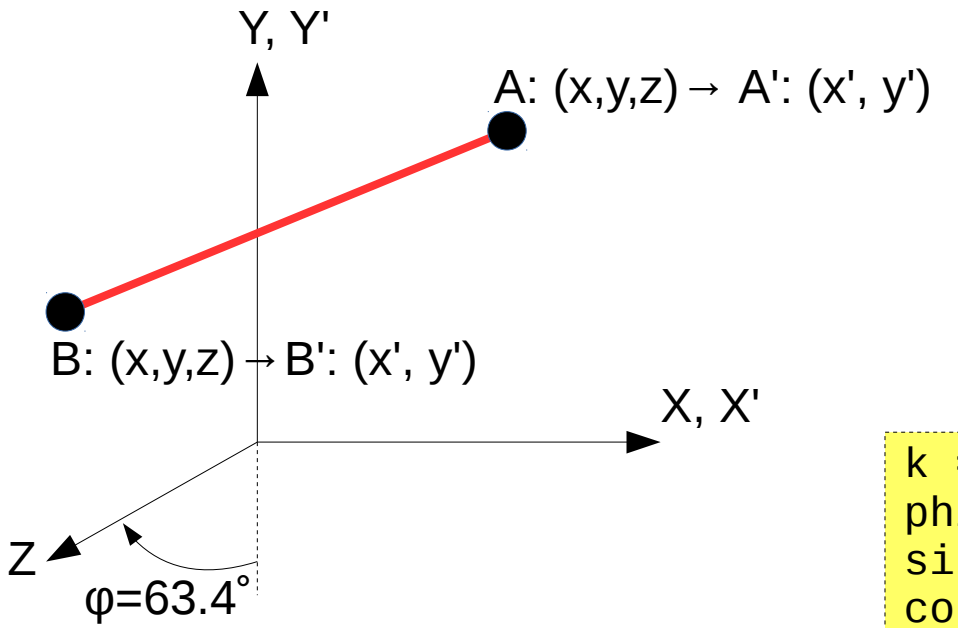
Perspective projection
- parallel lines **are not** preserved
- proportions **are not** preserved
- more realistic view
- complex computations



Parallel projection
- parallel lines **are** preserved
- proportions **are** preserved
- less realistic view (technical)
- easy computations



Cabinet (parallel) projection



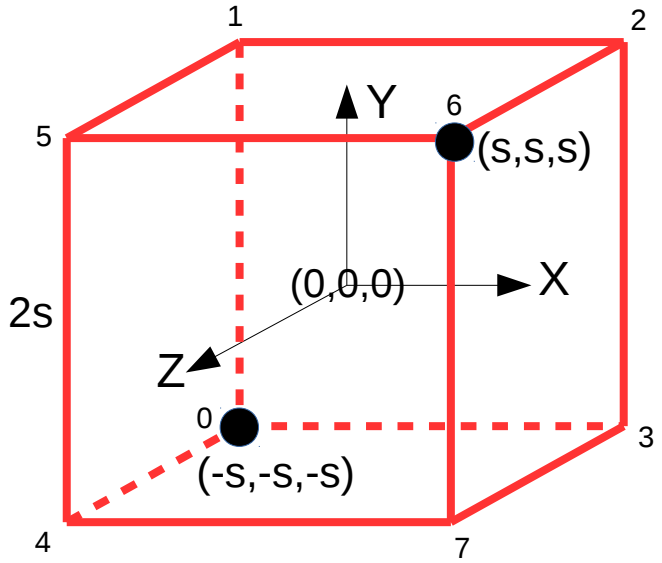
$$x' = x - z\sin(\phi)k$$
$$y' = y - z\cos(\phi)k$$

k - length scaling factor (= ½)

```
k = 0.5 # projection scale
phi = math.radians(63.4) # try also 45
sinphi = math.sin(phi)
cosphi = math.cos(phi)
```

```
def drawline3d(a, b, w = 1):    # a, b -> ap, bp
    xa, ya, za = a[0], a[1], a[2]
    xb, yb, zb = b[0], b[1], b[2]

    # convert 3d to 2d with cabinet projection
    xap = xa - za*sinphi *k
    yap = ya - za*cosphi *k
    xbp = xb - zb*sinphi *k
    ybp = yb - zb*cosphi *k

    # convert to pygame-screen coordinates and draw
    s1, s2 = (win.centerx+xap, win.centery-yap), (win.centerx+xbp, win.centery-ybp)
    pygame.draw.line(scr, black, s1, s2, w)
```

Cube



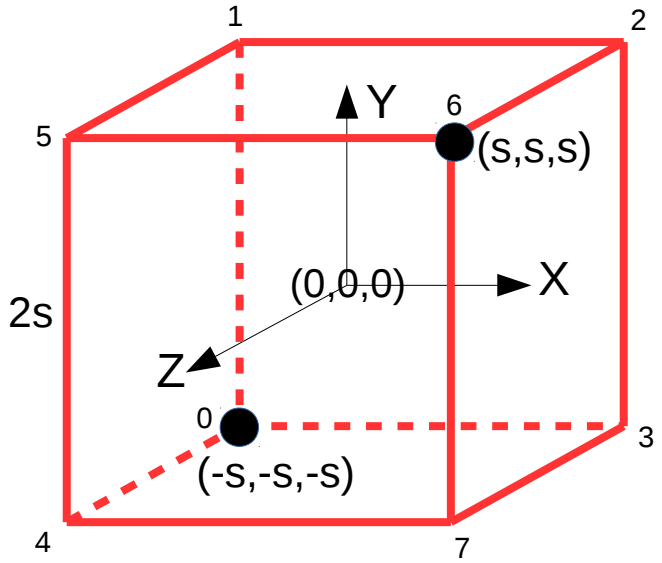
Origin of coordinate system (0,0,0) located in the center of the figure

Cube:

- 8 corners (points: P0 … P7),
- 12 edges (lines: Px, Py)
- 24 drawing points (pairwise)

```
s = 100

P0, P1, P2, P3 = (-s,-s,-s), (-s, s,-s), ( s, s,-s), ( s,-s,-s)
P4, P5, P6, P7 = (-s,-s, s), (-s, s, s), ( s, s, s), ( s,-s, s)


cube = (P0, P1,  P1, P2,  P2, P3,  P3, P0,
        P4, P5,  P5, P6,  P6, P7,  P7, P4,
        P0, P4,  P1, P5,  P2, P6,  P3, P7)
```

Cube

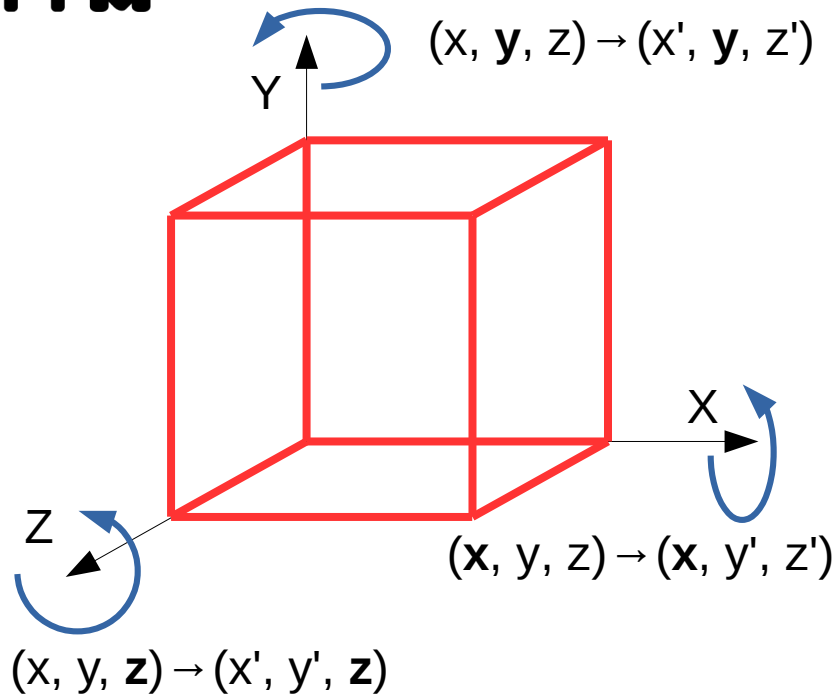



```
cube = ( P0,P1,   P1,P2,   P2,P3,   P3,P0,
         P4,P5,   P5,P6,   P6,P7,   P7,P4,
         P0,P4,   P1,P5,   P2,P6,   P3,P7 )
```

```
drawfigure3d(cube)
```

```
def drawfigure3d(figure):
    even = figure[0::2]
    odd  = figure[1::2]

    #read points in pairs
    for a, b in zip(even, odd):
        drawline3d(a, b)
```

3D rotation



Rotation in 3D space can be composed of three 2D rotations

$$
\begin{aligned}
x' &= x\cos(\alpha) - y\sin(\alpha) \\
y' &= x\sin(\alpha) + y\cos(\alpha) \\
z' &= z
\end{aligned}
$$

```
cube = rotx(cube,a)

cube = roty(rotx(cube,a),a)

cube = rotz(roty(rotx(cube,a),a),a)
```

```
def rotx(fig, a):
    newfig = []
    sin, cos = math.sin(a), math.cos(a)
    for x, y, z in fig:
        newfig.append((x,y*cos-z*sin,y*sin+z*cos))
    return newfig


def roty(fig, a):
    newfig = []
    sin, cos = math.sin(a), math.cos(a)
    for x, y, z in fig:
        newfig.append((x*cos+z*sin,y,-x*sin+z*cos))
    return newfig


def rotz(fig, a):
    newfig = []
    sin, cos = math.sin(a), math.cos(a)
    for x, y, z in fig:
        newfig.append((x*cos-y*sin,x*sin+y*cos,z))
    return newfig
```

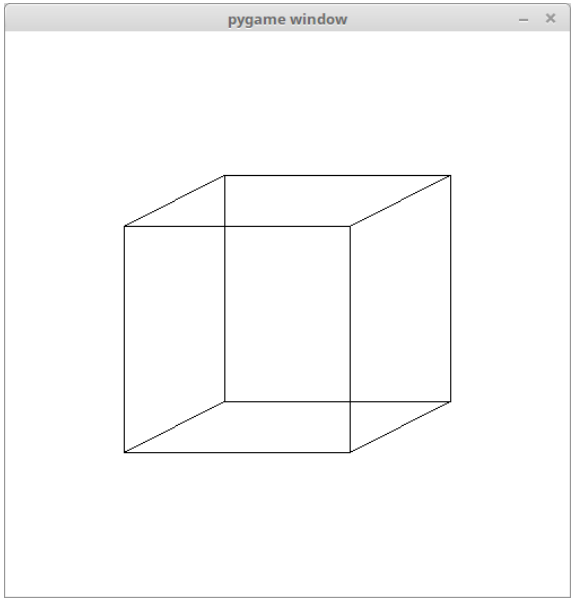
Program structure

cube3d.py

initialization

```
def drawline3d(...)
def draw3d(...)
def rotx(...)
def roty(...)
def rotz(...)

constants

cube definition
```

pygame loop



initialization

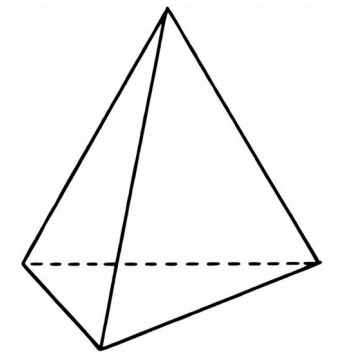
```
import pygame, sys, math
black, white = (0, 0, 0), (255, 255, 255)

pygame.init()
scr = pygame.display.set_mode((500, 500))
win = scr.get_rect()
scr.fill(white)
```

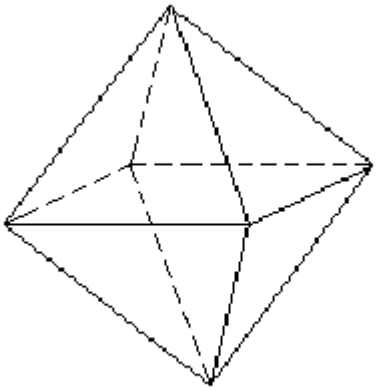
pygame loop

```
pygame.key.set_repeat(50,50)

while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            sys.exit()
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_x:
                cube = rotx(cube,angle)
            if event.key == pygame.K_y:
                cube = roty(cube,angle)
            if event.key == pygame.K_z:
                cube = rotz(cube,angle)

    scr.fill(white)
    drawfigure3d(cube)
    pygame.display.flip()
```

Exercise 1

- Define and display:
  - tetrahedron
  - octahedron

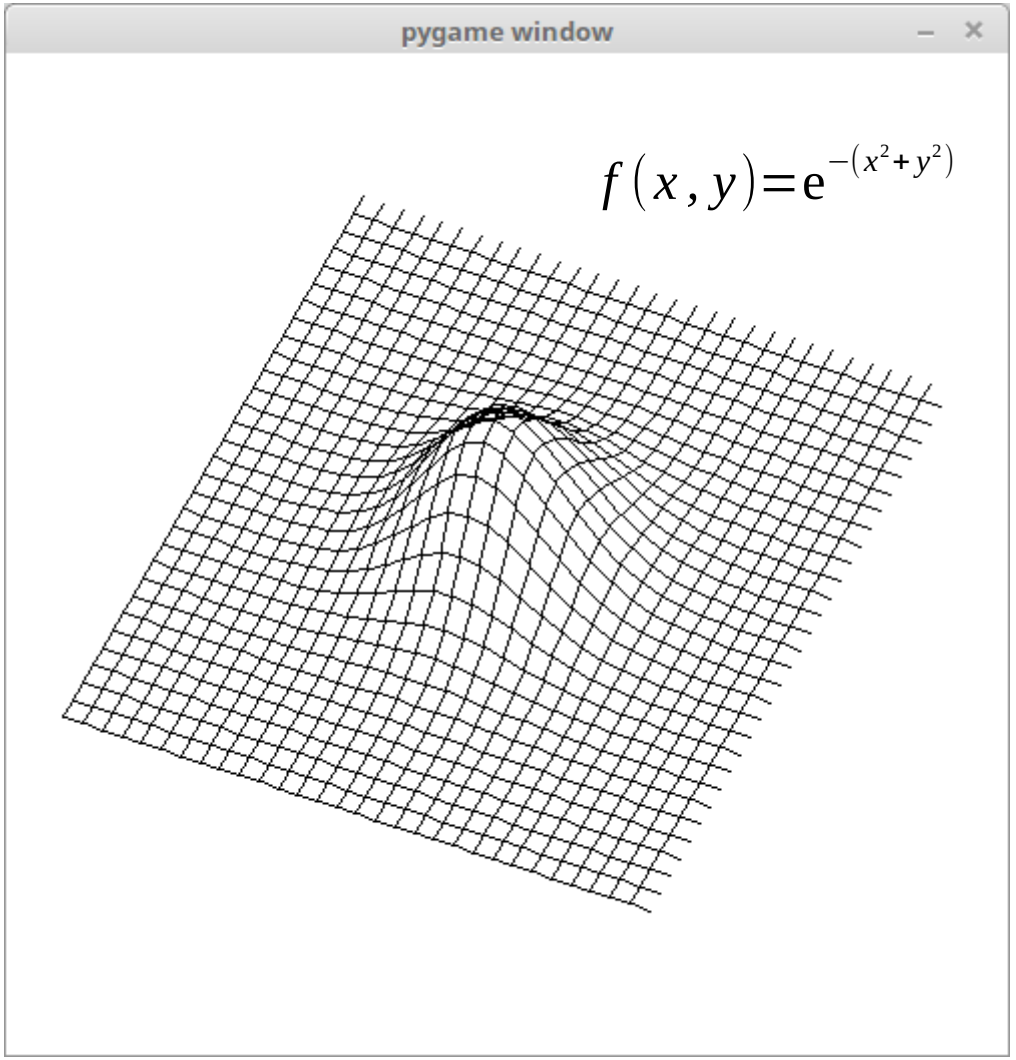




- Optionally:
  - Implement backward rotation
  - Implement zoom
  - Implement translations (directional shifts)

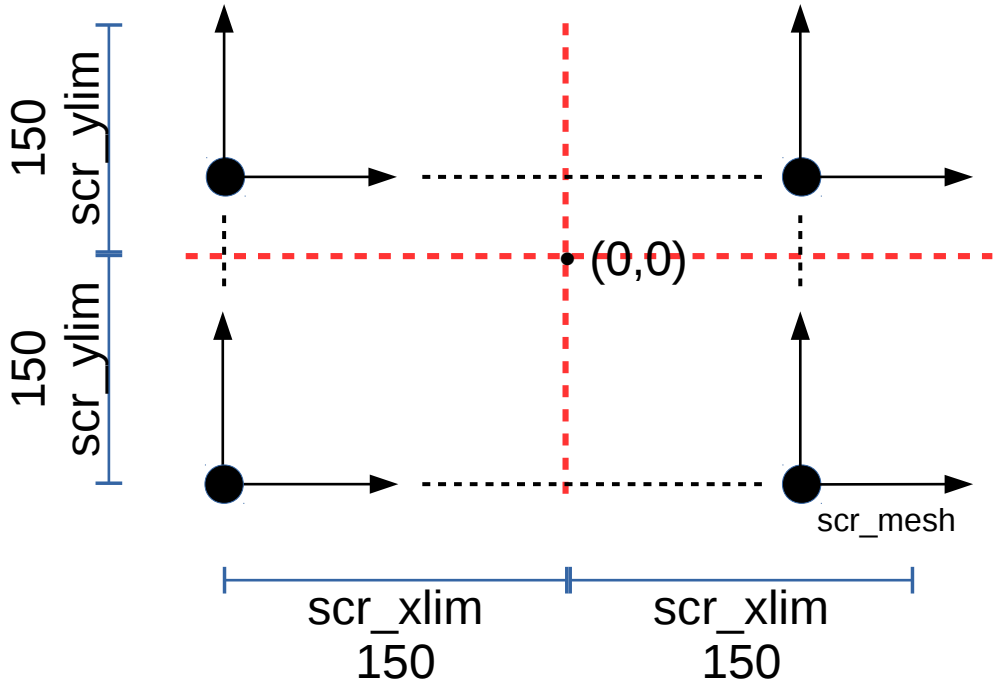
3D function viewer

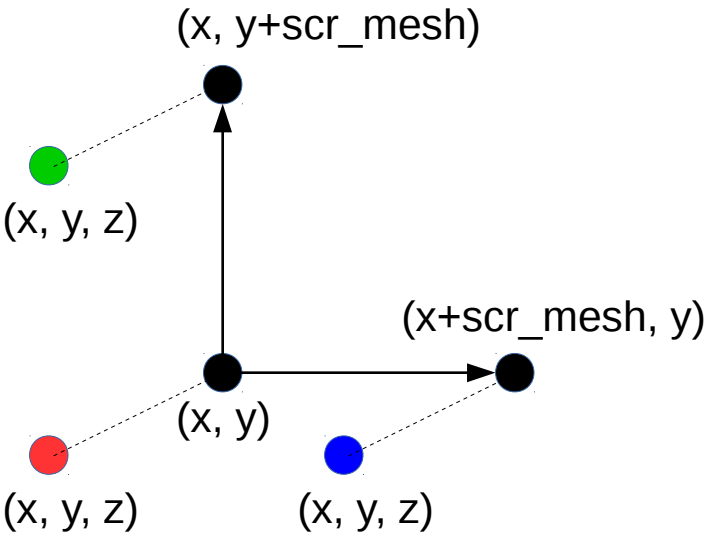
- define z = f(x,y)
- compute all (x,y,z) points and store 3D wire-mesh as segment-lines
- draw the wire-mesh with drawfigure3D()




```
z = math.exp(-((x)**2 + (y)**2))
```

Making (x,y,z) wire-mesh

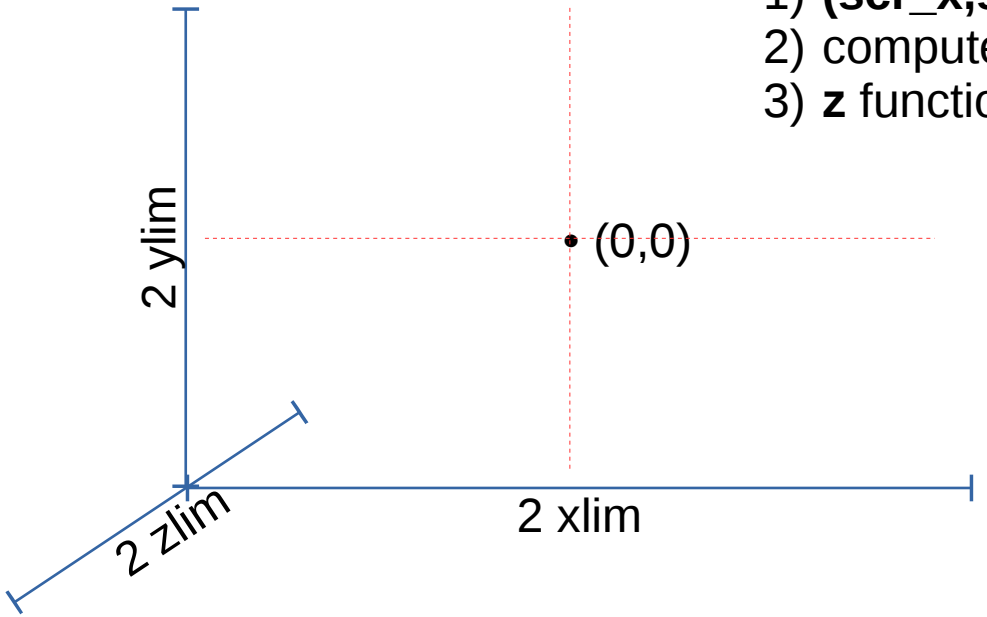





```
# drawing limits for screen - around (0,0)
scr_xlim, scr_ylim, scr_zlim = 150, 150, 150
scr_mesh = 10

# create function mesh
wiremesh = []
for x in range(-scr_xlim, scr_xlim, scr_mesh):
    for y in range(-scr_ylim, scr_ylim, scr_mesh):
        wiremesh.extend([(x,y,z(x,y)), (x+scr_mesh,y,z(x+scr_mesh,y))])
        wiremesh.extend([(x,y,z(x,y)), (x,y+scr_mesh,z(x,y+scr_mesh))])
```

Computing z(x, y)

1) **(scr_x,scr_y)** screen coord. → **(x,y)** function coords.
2) compute **z = (x,y)**
3) **z** function coords → **scr_z** screen coord.



$$x = scr\_x \frac{xlim}{scr\_xlim}$$

$$y = scr\_y \frac{ylim}{scr\_ylim}$$

$$scr\_z = z \frac{scr\_zlim}{zlim}$$

```
# drawing limits for 3d space - around (0,0,0)
xlim, ylim, zlim = math.pi, math.pi, 1.0

def z(scr_x,scr_y):
    x = (float(scr_x) / scr_xlim) * xlim
    y = (float(scr_y) / scr_ylim) * ylim

    z = math.exp(-((x)**2 + (y)**2))

    return (z / zlim) * scr_zlim
```

Exercise 2

wiremesh3d.py

- Define and draw other 3D functions
- Optionally:
  - Draw the direction axes
  - Show the bounding box for drawing

initialization

```
def drawline3d(...)
def drawfigure3d(...)
def rotx(...)
def roty(...)
def rotz(...)
def z(...):

constants

make wire-mesh
```

pygame loop