1) PyGame vs PyOpenGL

PyGame handles the application framework: creating the window (pygame.display.set_mode), reading input, timing, and generating text textures via font.render() and pygame.image.tostring(). PyOpenGL performs the actual 3D rendering using OpenGL calls such as glBegin, glVertex3f, glLightfv, and gluPerspective. In short, PyGame prepares data and interaction, while PyOpenGL sends geometry and state to the GPU. They work together: PyGame manages the environment; PyOpenGL draws the scene.

Why Pillow (PIL)?

OpenGL cannot decode PNG/JPG files, it only accepts raw pixel arrays. Pillow loads images using Image.open().convert() and converts them to bytes for glTexImage2D. It also ensures correct color format and handles vertical flipping before upload. Without PIL, you would need your own image decoder before texturing.

2) GL_PROJECTION vs GL_MODELVIEW

GL_PROJECTION defines the camera lens (perspective volume and clipping planes). GL_MODELVIEW controls object placement and camera movement using transforms like glTranslatef() and glRotatef(). Separating them keeps camera geometry independent from object transformations.

Where gluPerspective() belongs

gluPerspective() must be called while glMatrixMode(GL_PROJECTION) is active because it defines the viewing frustum. Calling it under GL_MODELVIEW would distort objects instead of changing perspective. Therefore set_projection() switches to projection mode, applies gluPerspective, then switches back.

3) Inside load_texture()

glTexImage2D() uploads pixel data to GPU memory as a texture.
GL_TEXTURE_WRAP_S/T = GL_REPEAT lets coordinates beyond [0,1] tile across the floor instead of stretching. glTexParameteri(... GL_LINEAR) smooths scaling when the texture is magnified or reduced.

Why flip vertically (Image.FLIP_TOP_BOTTOM)?

Image files store pixels top-to-bottom, but OpenGL texture coordinates start bottom-to-top.
Without flipping, (0,0) samples the wrong corner and textures appear upside-down.
Flipping aligns image coordinates with OpenGL's texture system.

4) Role of GL_LIGHT0

GL_LIGHT0 is the main scene light providing ambient, diffuse, and specular shading via glLightfv. Objects respond to it using normals and material properties. Without it, geometry appears flat and uniformly colored.

Why update position every frame

glLightfv(GL_LIGHT0, GL_POSITION, light_pos) is transformed by the current modelview

matrix.
Since the camera moves each frame, the light would otherwise move with the camera. Updating it each frame keeps the light fixed in world coordinates.

5) Why culling hides the floor
GL_CULL_FACE removes polygons facing away from the camera based on vertex winding order. A single quad floor only has one visible side, so when viewed from below it becomes a back face and disappears. This is expected behavior in single-sided geometry.

How the code prevents it
The program disables culling for overlay/floor drawing using glDisable(GL_CULL_FACE). This forces OpenGL to draw both sides of the quad. Thus the floor remains visible regardless of camera angle.

6) Why draw transparent objects last
Opaque objects write depth values first using GL_DEPTH_TEST. Transparent objects must blend with already-drawn colors, so they are rendered afterward. Otherwise blending results become incorrect.

Purpose of glDepthMask(GL_FALSE)
glDepthMask(GL_FALSE) prevents the transparent sphere from writing to the depth buffer.
It still depth-tests but doesn’t block objects behind it. This allows correct blending when using glEnable(GL_BLEND) and glBlendFunc.