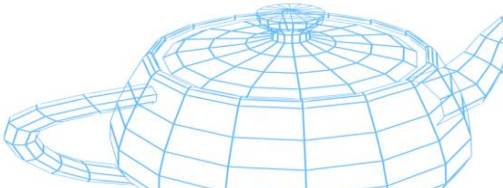
SUPSI

Computer Graphics

3D Graphics Engines (2): advanced architecture

Achille Peternier, adjunct professor



3D graphics engine – additional components

Light

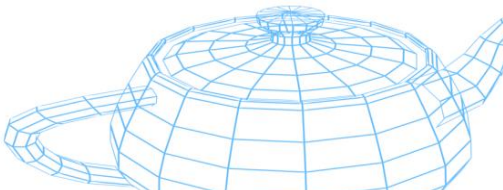
Light class that implements the main types of light introduced in the course. This class includes the necessary methods for applying its settings to OpenGL.

Material

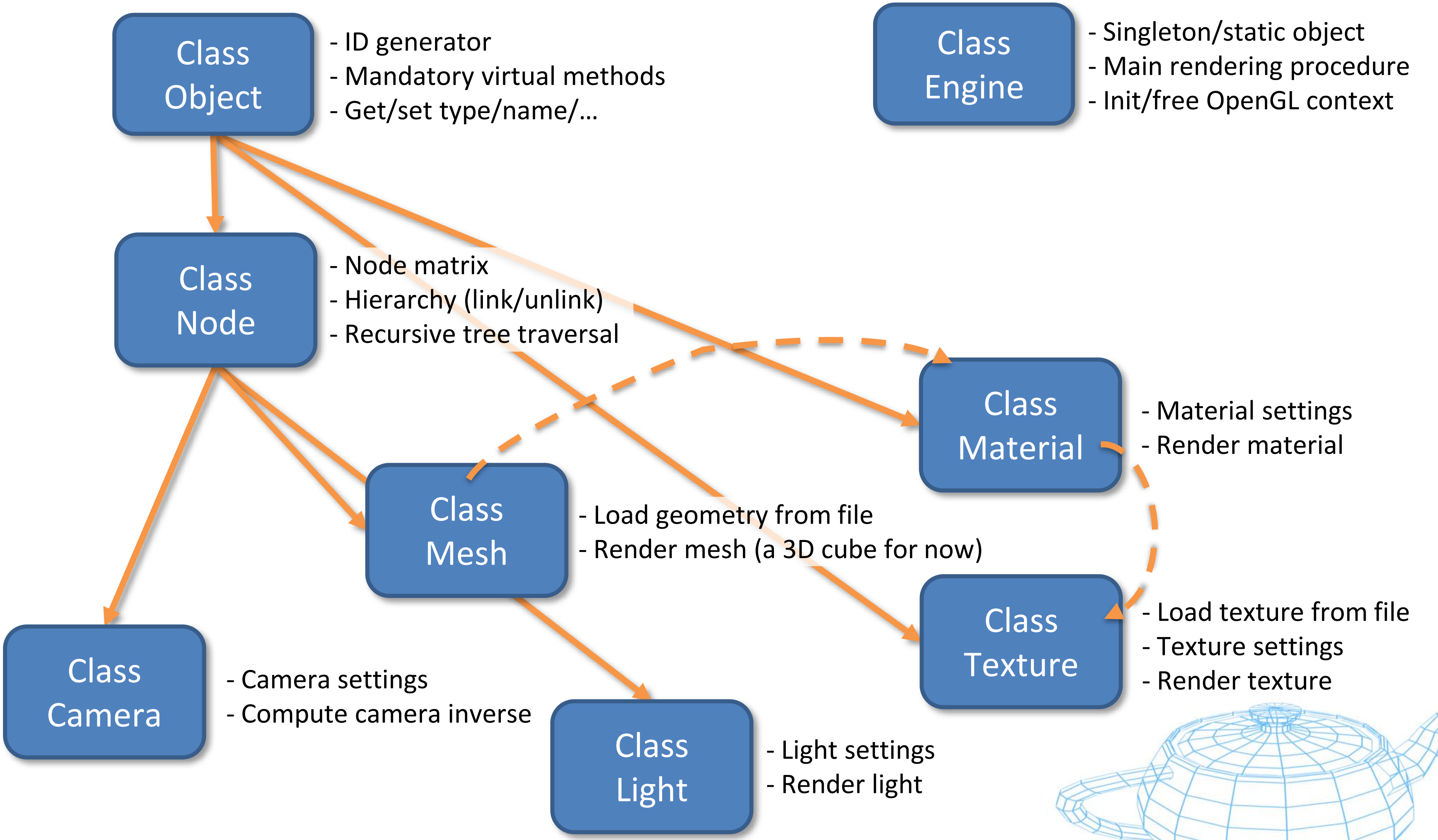
Contains all the parameters to define a material. It enables changing material properties, and it is responsible for transferring its settings to OpenGL through the necessary methods.

Texture

This class represents a texture. It is responsible for loading data from a file into an OpenGL texture and for passing its settings to the OpenGL API. *(More about in the OpenGL 4 chapter)*



3D graphics engine – refined architecture



Class Object

- ID generator
- Mandatory virtual methods
- Get/set type/name/…

Class Engine

- Singleton/static object
- Main rendering procedure
- Init/free OpenGL context

Class Node

- Node matrix
- Hierarchy (link/unlink)
- Recursive tree traversal

Class Material

- Material settings
- Render material

Class Mesh

- Load geometry from file
- Render mesh (a 3D cube for now)

Class Texture

- Load texture from file
- Texture settings
- Render texture

Class Camera

- Camera settings
- Compute camera inverse

Class Light

- Light settings
- Render light

Scene graph example

🟦 node

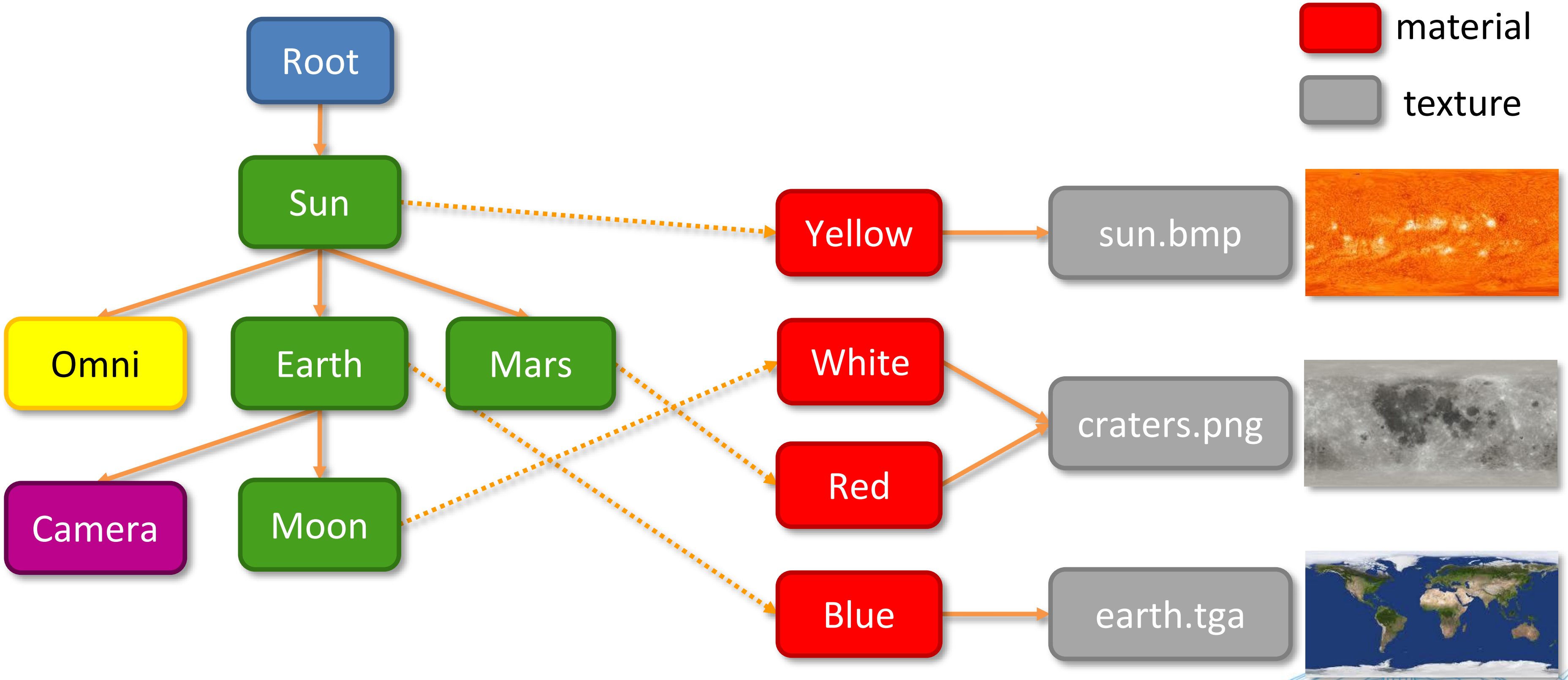
🟨 light

🟩 mesh

🟪 camera

🟥 material

⬜ texture



Root

Sun

Omni

Earth

Mars

Camera

Moon

Yellow

White

Red

Blue

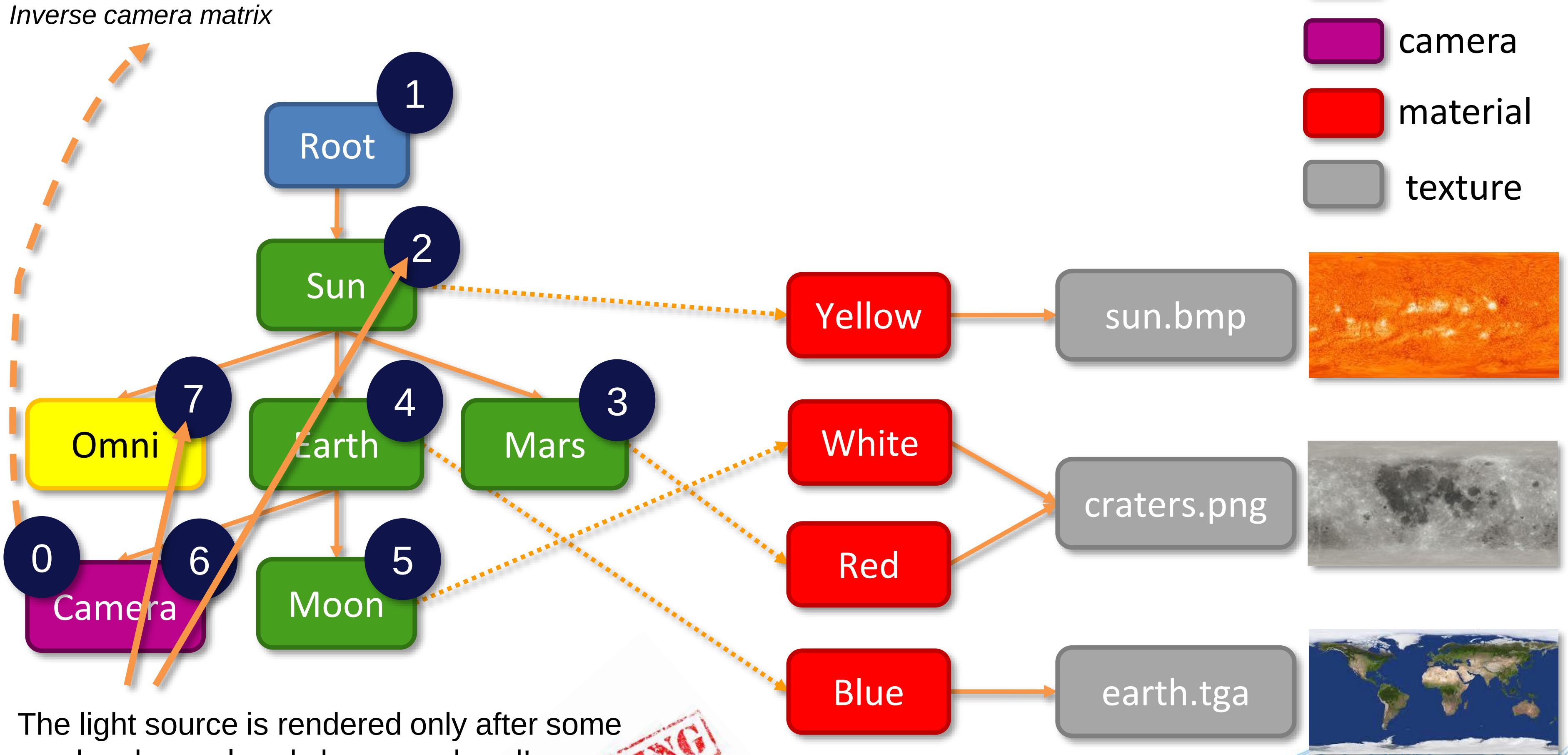
sun.bmp

craters.png

earth.tga

Scene graph example: rendering order

- 🟦 node
- 🟨 light
- 🟩 mesh
- 🟪 camera
- 🟥 material
- ⬜ texture



The light source is rendered only after some meshes have already been rendered!

WARNING

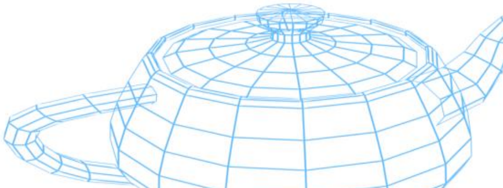
Indirect rendering

- Do not render the nodes directly while you traverse the scene graph, but:

1) Create a list and store all the objects to render with their final matrix (in world coordinates):
   - Each list’s entry is (at least) an **`object*`** and **`glm::mat4`** pair.

2) Sort objects within the list as needed (e.g., lights first, then meshes).

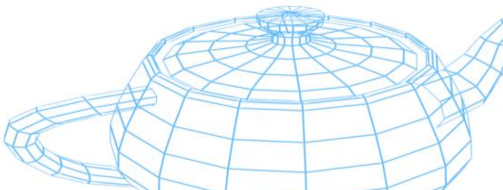
3) Iterate through the list and call the **`render()`** method of each object:
   - Each matrix in the list is multiplied by the inverse of the camera matrix.
   - You can re-render the same scene from different points of view without refreshing the list’s entries.



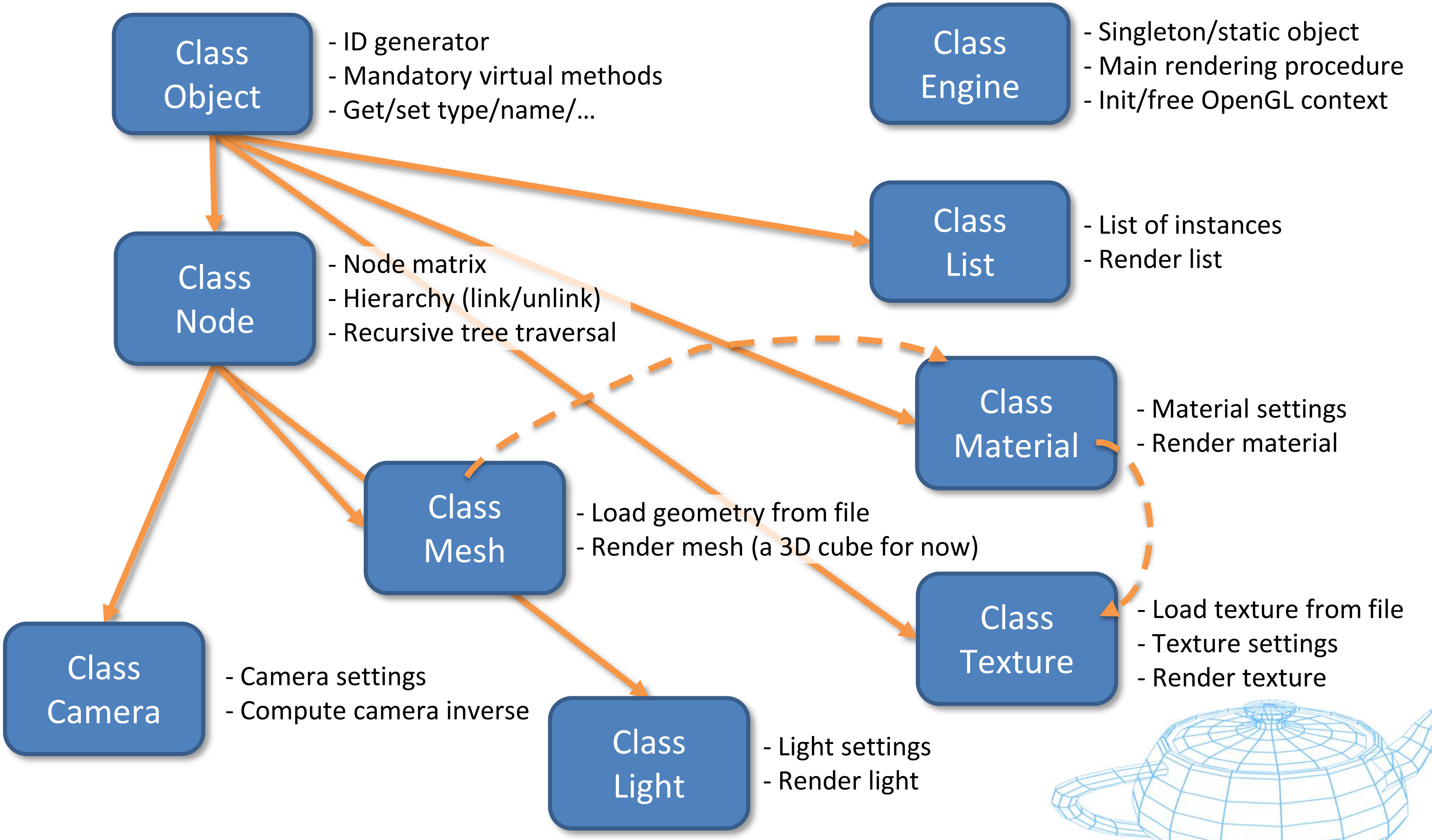
3D graphics engine – additional components

List

Contains a list of (pointers/references to) object instances, each one with its own properties (such as position, material, etc.). Matrices are stored in world coordinates after being evaluated according to their scene graph hierarchy.



3D graphics engine – refined architecture



Class Object

- ID generator
- Mandatory virtual methods
- Get/set type/name/…

Class Engine

- Singleton/static object
- Main rendering procedure
- Init/free OpenGL context

Class List

- List of instances
- Render list

Class Node

- Node matrix
- Hierarchy (link/unlink)
- Recursive tree traversal

Class Material

- Material settings
- Render material

Class Mesh

- Load geometry from file
- Render mesh (a 3D cube for now)

Class Texture

- Load texture from file
- Texture settings
- Render texture

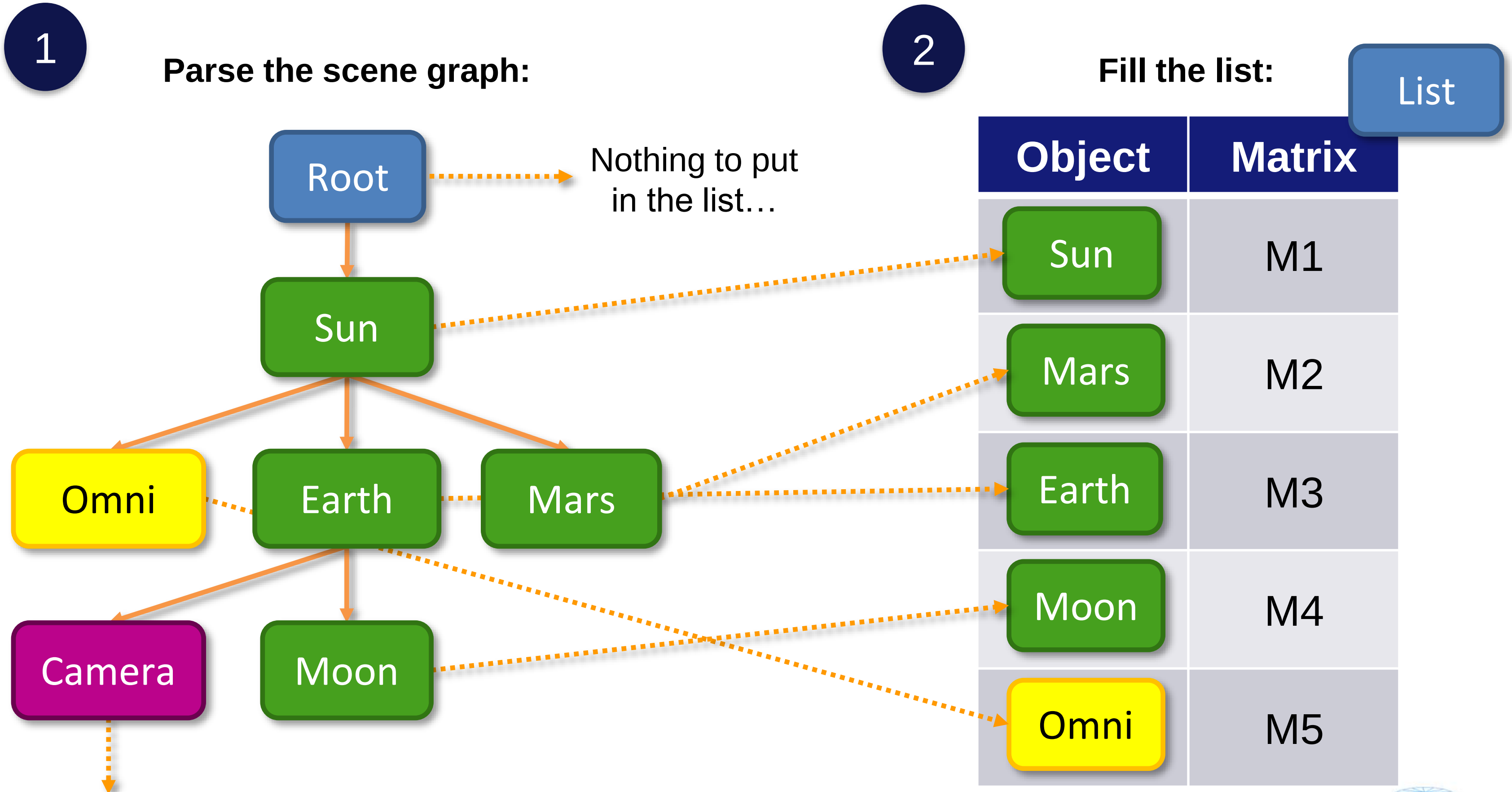
Class Camera

- Camera settings
- Compute camera inverse

Class Light

- Light settings
- Render light

Indirect rendering



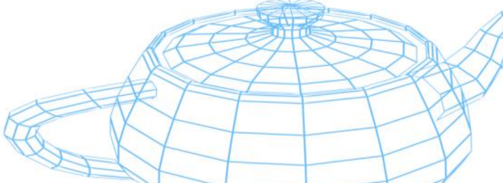
1 **Parse the scene graph:**

2 **Fill the list:**

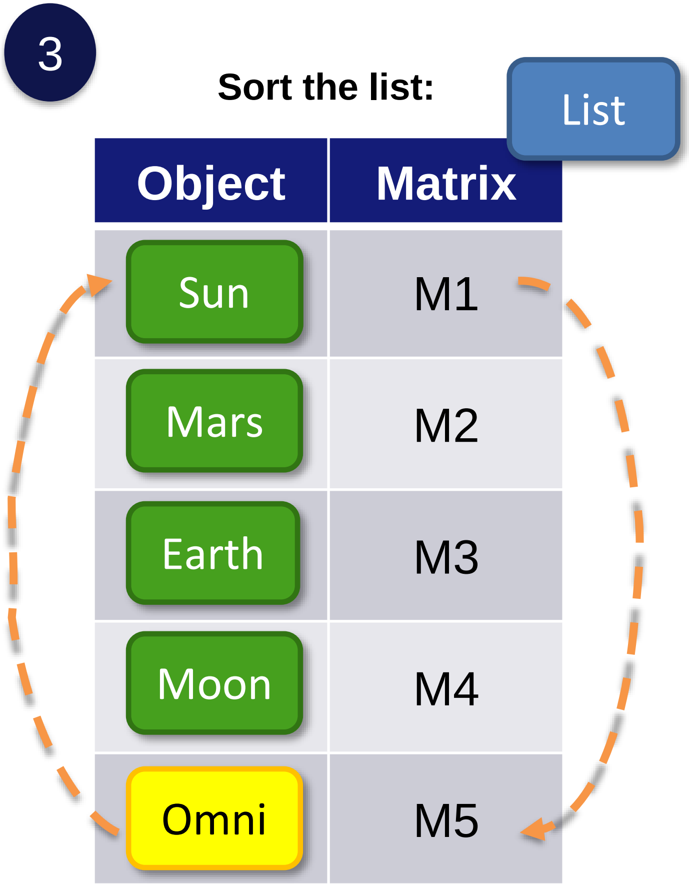
Nothing to put in the list…

Object	Matrix
Sun	M1
Mars	M2
Earth	M3
Moon	M4
Omni	M5

Nothing to put in the list (but store the final camera matrix somewhere)



Indirect rendering



3

Sort the list:

List

Object	Matrix
Sun	M1
Mars	M2
Earth	M3
Moon	M4
Omni	M5

Camera

C = final camera matrix

4

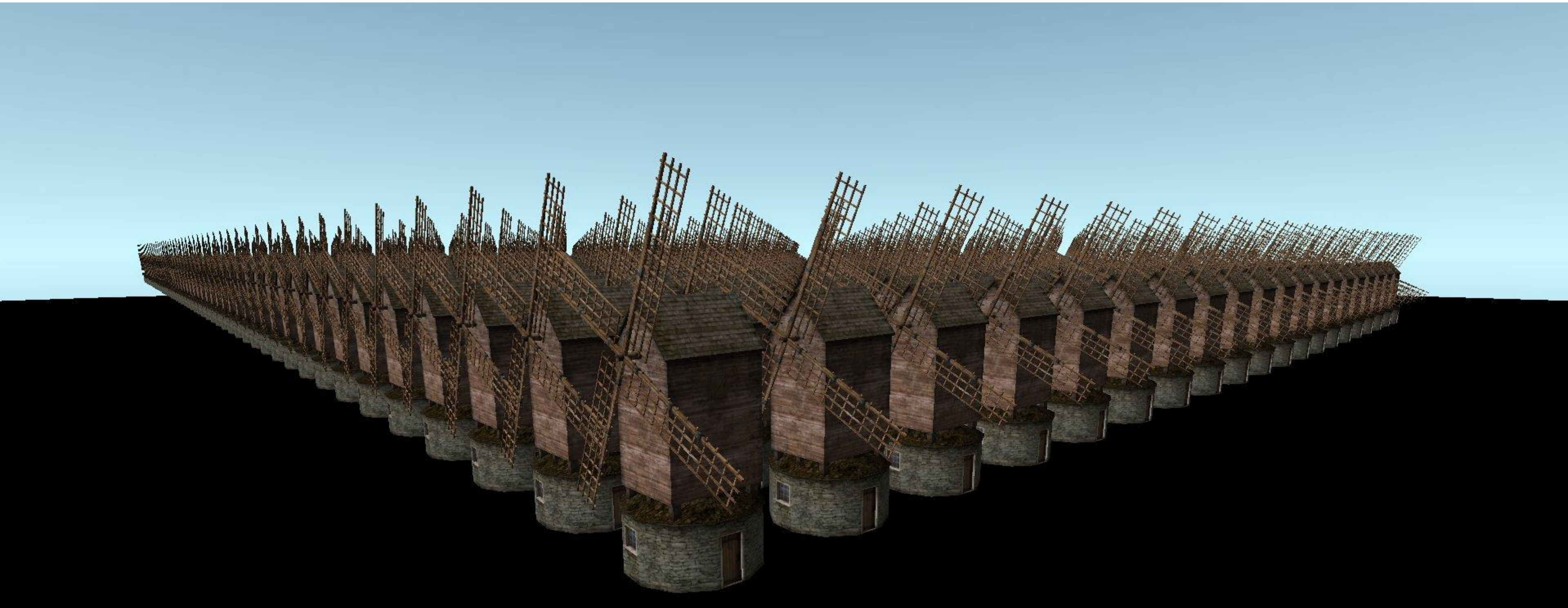
Render the list:

List

Object	Matrix	
Omni	M5	list[0].obj->render(C^{-1} * list[0].matrix);
Mars	M2	list[1].obj->render(C^{-1} * list[1].matrix);
Earth	M3	list[2].obj->render(C^{-1} * list[2].matrix);
Moon	M4	list[3].obj->render(C^{-1} * list[3].matrix);
Sun	M1	list[4].obj->render(C^{-1} * list[4].matrix);

Instancing

- The list class can be also used to render one same element (mesh, light, etc.) multiple times at different coordinates and/or using different parameters:
  - Simply traverse and queue to the same list one same scene graph (or portions of) with different node matrices.



Instancing

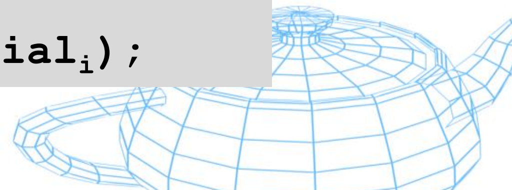
Code example:

```
planet.setMatrix(matrixA);
planet.setMaterial(blue);
list.pass(planet);

planet.setMatrix(matrixB);
planet.setMaterial(red);
list.pass(planet);

planet.setMatrix(matrixC);
planet.setMaterial(blue);
list.pass(planet);

Engine.render(camera, list);
```

List of objects to render:

Object	Matrix	Material
Planet	M1	Blue
Planet	M2	Red
Planet	M3	Blue

```
for i = each object of the list
   Object$_i$->render(camera$^{-1}$ * Matrix$_i$, Material$_i$);
```



Instancing

- The **`pass()`** method should be recursive and parse all the child nodes linked to the node:
  - At each recursion, invoke the **`pass()`** method of the child nodes:
    - The child node matrix is multiplied by the parent node’s final matrix.
    - Use an internal push/pop mechanism to keep track of the current matrix.

- In this way, invoking the **`pass()`** method on the root node will add the content of the entire scene to the list :
  - The list will contain all the scene objects in world coordinates.
  - Multiple scene graphs (or the same one processed multiple times) can be queued to the same list.
  - Multiple lists can be used (e.g., one for 2D and one for 3D rendering).