# Assignment 3+4 Report

Shobhit Behl (IMT2016024)

- Problem Statement 1

  Rendering textures and implementing texture maps on different geometrical models.

  Objects -
  1.) Cube
  2.) Cylinder
  3.) Sphere
  4.) Beethoven (Object with arbitrary geometry)

  Textures -
  1.) Wood
  2.) Checkerboard
  3.) World Map
  4.) A Face

  Mappings -
  1.) Planar
  2.) Cylindrical
  3.) Spherical

  Try out all combinations of models, textures and mappings.

- Problem Statement 2

  Implement a scene graph and animate the 4 objects used in the previous problem and vary speed of movement.

  1.) Object A is fixed with respect to the floor.

  2.) Object B moves in a circular (or similar path) around object A, and wobbles left and right as it traverses this path.

  3.) Object C starts some distance away from B, and tries to move towards B, constantly adjusting its direction depending on the current position of B. The speed of C is roughly half that of B.

4.) Object D sits on top of C, and jumps up and down while staying on top of C.

- Design

1.) *Parser Class* - This class is used to read ply files and return the number of faces, number of vertices, the vertices, the element index array. and certain information pertaining to selection logic.

2.) *Shader Class* - This is used to create, add and compile shaders and get an identifier which can be used to specify the shader programs to be used.

3.) *Texture Class* - This class is used to load the texture onto the GPU and bind it to a slot so the shaders can access it.

4.) *Vertex Class* - This class is used to store various attributes of each vertex which is used in the shaders i.e. position, colour, normal etc.

5.) *Model Class* - This class represents a geometric model, i.e. a mesh. It has an array of vertices, an array of indexes, transformation matrices, vertex array identifier, vertex buffer identifier and an index buffer identifier. It also has extra information pertaining to selection of the model to check and store whether the model has been clicked or not.
It has a vector of model pointers which are it's children, these are used for making the scene graph.
It has a construct method which takes in a file name, creates a parser object and reads the file to get the vertices, indexes etc.
It also does the calculation of normals, splatting and the displaying of the model on the window.
There are methods to handle any user input and perform transformations by modifying the transformation matrices.
It also contains methods defining the texture maps.

6.) *Scene Class* - This class corresponds to the Model in MVC, it has an array of geometric models, methods to add models and perform transformations in the models.

7.) *View Class*- This class corresponds to View in MVC, it handles the creation of windows and initialization of various OpenGL requirements.

8.) *Controller Class* - This class corresponds to the Controller in MVC, it has a view and a scene object. It handles any interaction between the user and the View/Scene, displaying of the scene and also creates a Shader object to compile shader programs. It also loads and binds textures to the GPU.

- Logic

1.) *Scene Graph* - The scene graph is modelled using a vector of $Model$ pointers within $Model$ class which are it's children. A method which is called on the parent is also recursively called onto the children thus performing a dfs on the scene graph.

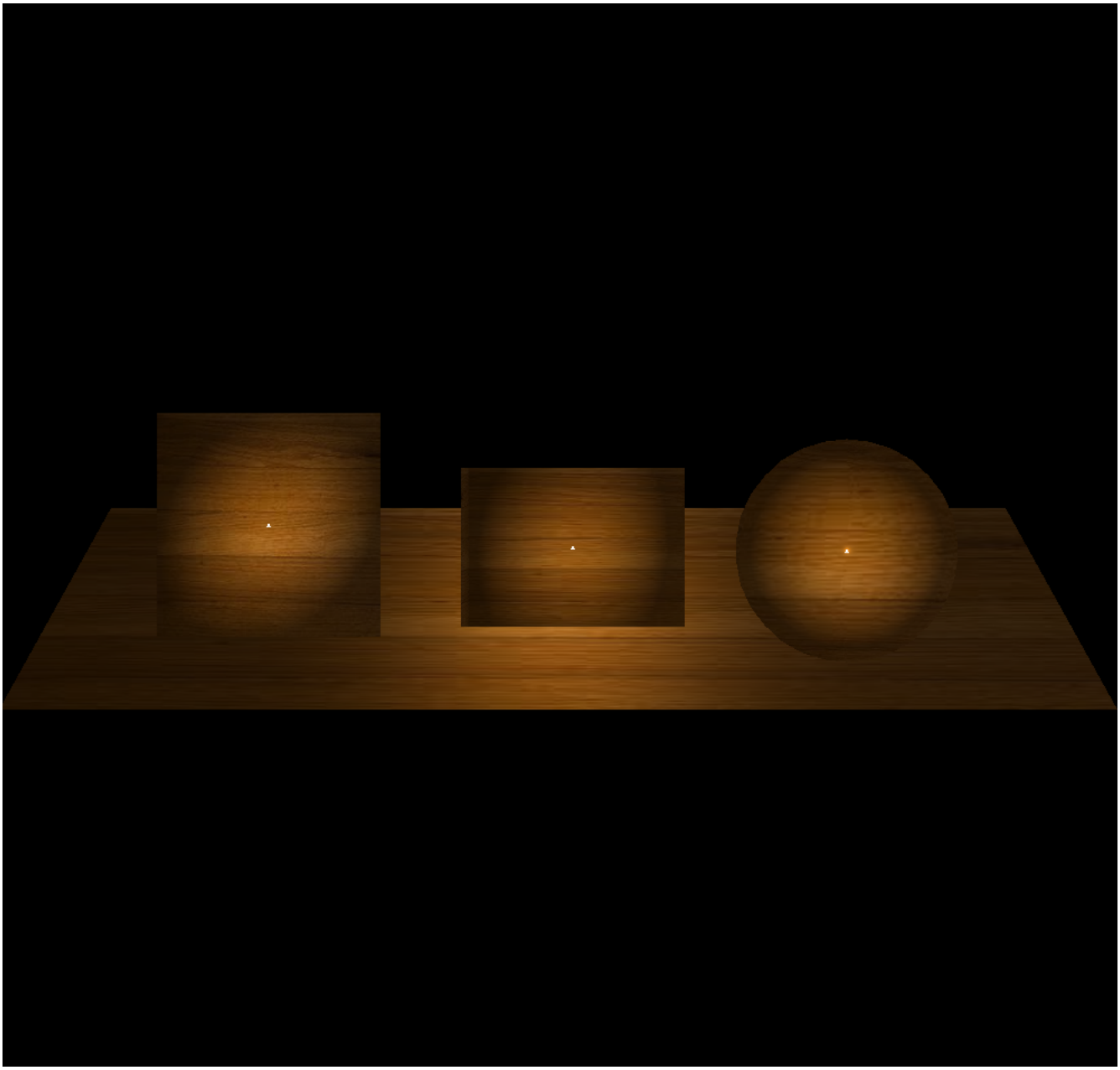
2.) *Planar Texture* - We ignore the z coordinate of the points and map x and y coordinates of the model to the texture.

3.) *Cylindrical Texture* - We map the y coordinate of the models to the y coordinate of the texture but for x coordinate mapping we use angle $\theta$ which goes around the curved surface of the cylinder.

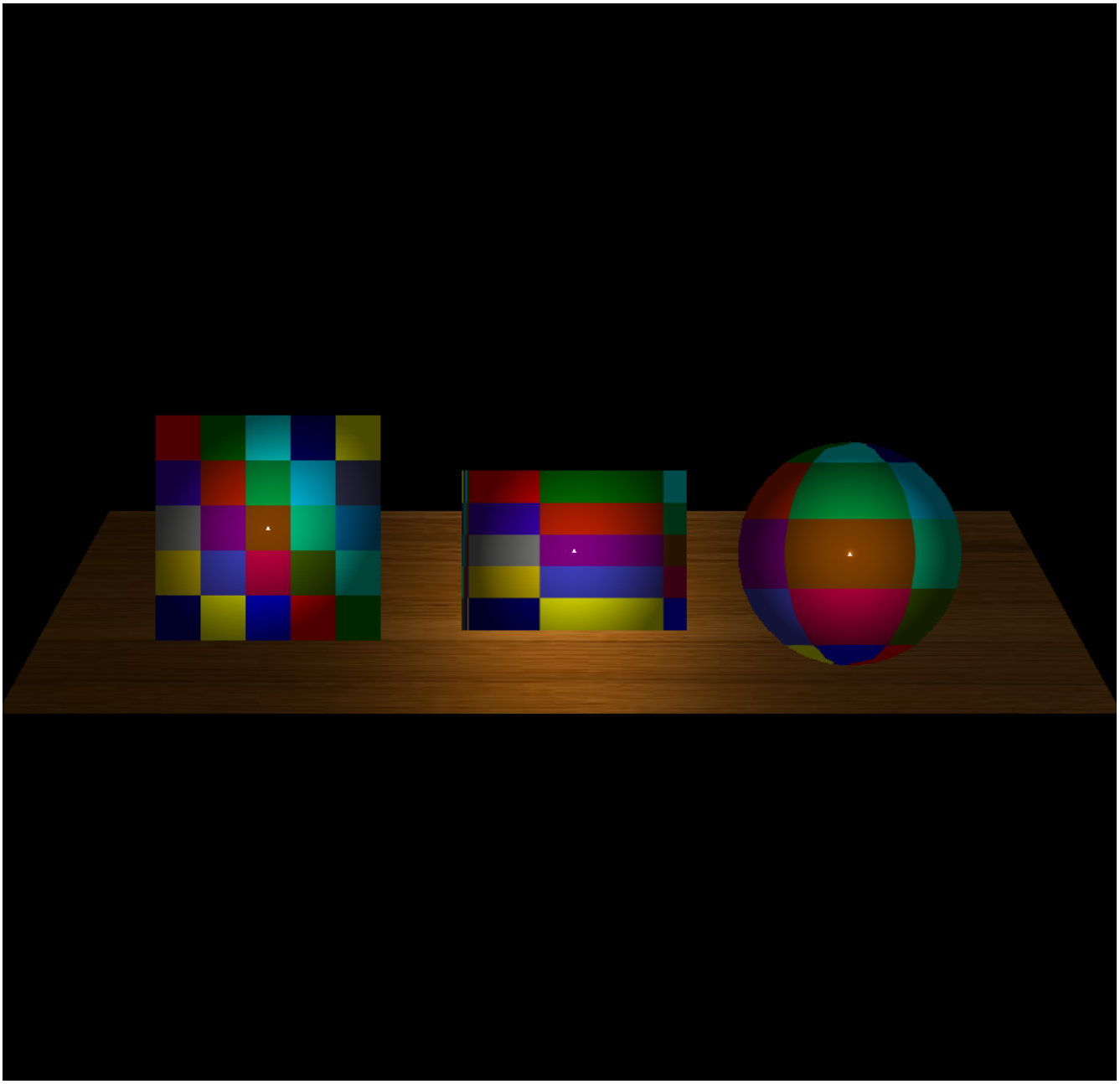
4.) *Spherical Texture* - We map y coordinate to angle $\phi$ which goes around the curved surface in the y-z plane and the x coordinate to angle $\theta$ which is similar to that in the cylindrical texture map and goes around the x-z plane.

5.) *Animation* - Object A is the root of the scene graph tree, Object B is a child of object A, Object C is a child of object B and object D is a child of Object C. The motion is controlled using an integer which determines the type of motion the object follows.
To make the light follow the object, the Model class now stores the light position associated to it and all its transforms are also applied to the light.
The transforms which one wants to apply to the children of a parent can be controlled by controlling the argument $worldMatrix$ passed in the display recursive call to the child.

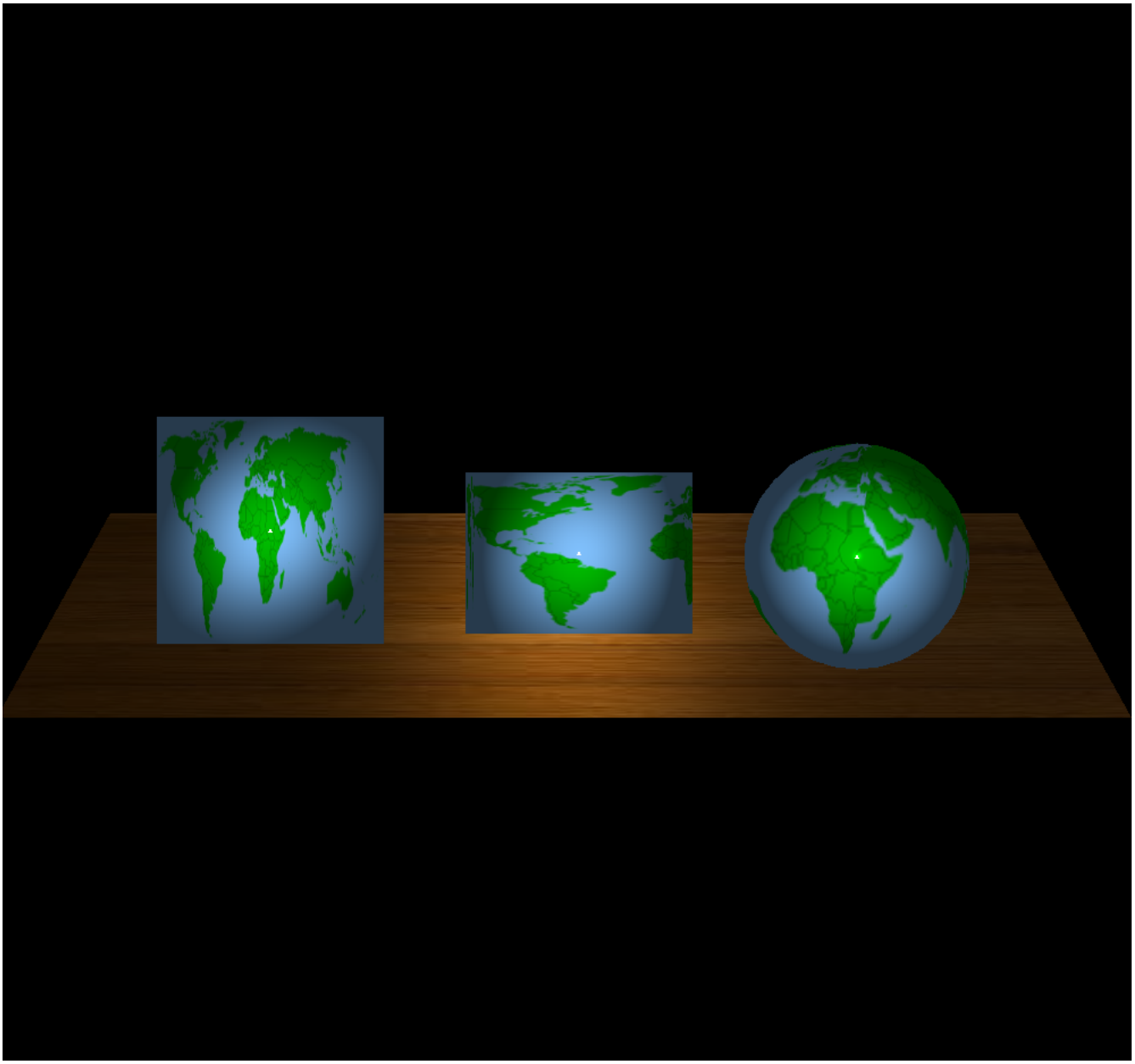
- Images



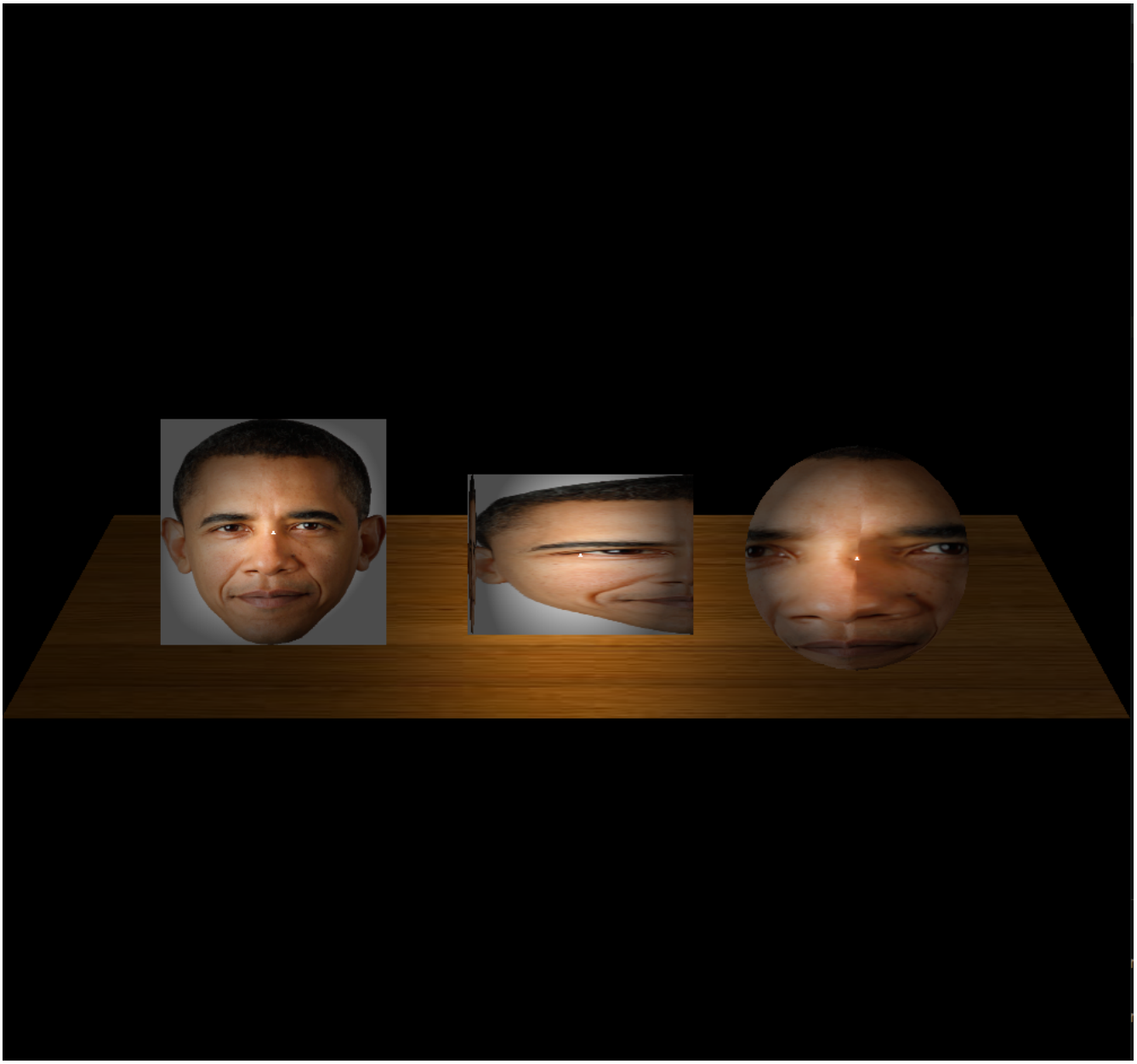
Wood texture using planar, cylindrical and spherical mapping from left to right



Checkerboard texture using planar, cylindrical and spherical mapping from left to right



World Map texture using planar, cylindrical and spherical mapping from left to right



Face texture using planar, cylindrical and spherical mapping from left to right