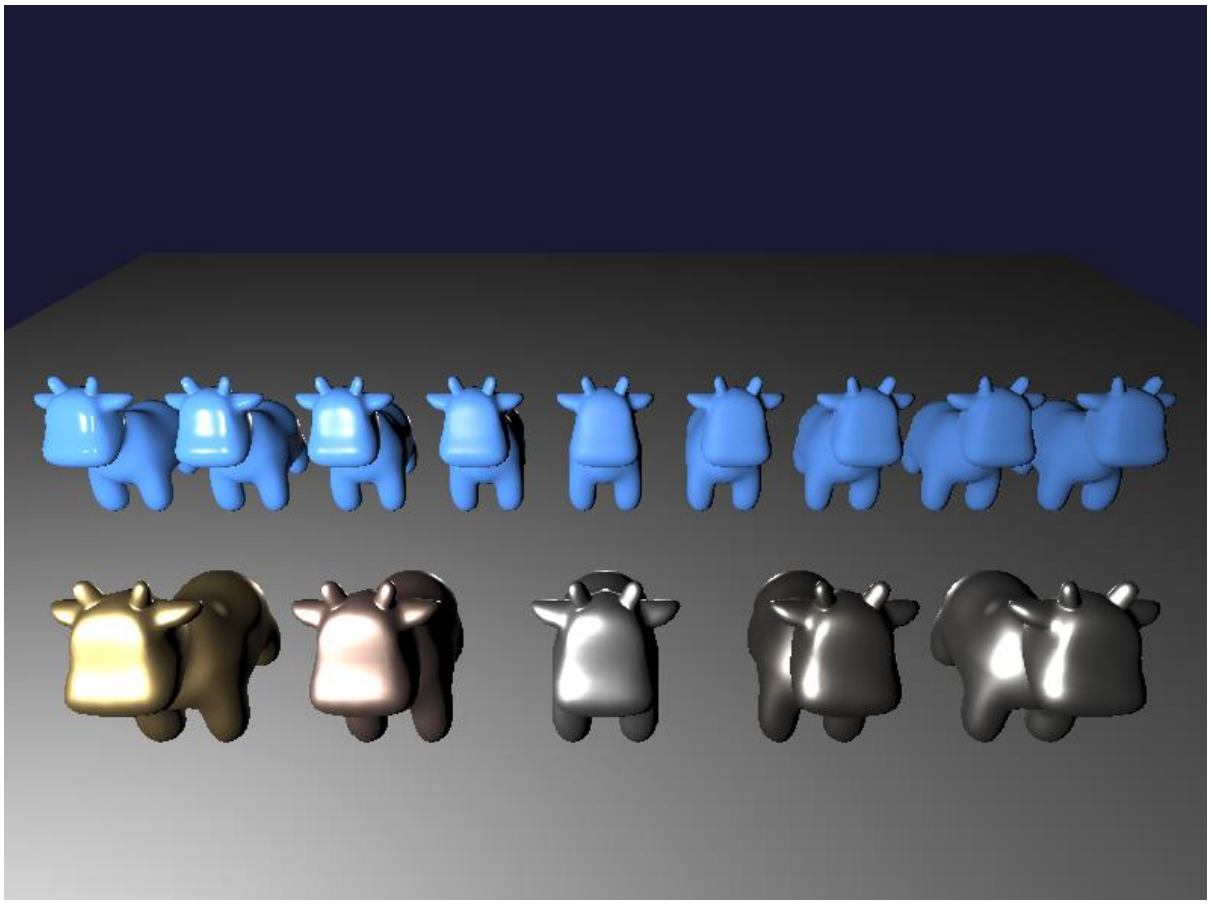
PBR

The end result:



Vertex shader:

```
layout (location = 0) in vec3 VertexPosition;
layout (location = 1) in vec3 VertexNormal;

out vec3 Position;
out vec3 Normal;

uniform mat4 ModelViewMatrix;
uniform mat3 NormalMatrix;
uniform mat4 MVP;

void main() {
    Normal = normalize( NormalMatrix * VertexNormal);
    Position = ( ModelViewMatrix * vec4(VertexPosition,1.0) ).xyz;

    gl_Position = MVP * vec4(VertexPosition,1.0);
}
```

Fragment shader:

```
const float PI = 3.14159265358979323846;

in vec3 Position;
in vec3 Normal;

uniform struct LightInfo {
  vec4 Position;  // Light position in cam. coords.
  vec3 L;         // Intensity
} Light[3];

uniform struct MaterialInfo {
  float Rough;     // Roughness
  bool Metal;      // Metallic (true) or dielectric (false)
  vec3 Color;      // Diffuse color for dielectrics, f0 for metallic
} Material;

layout( location = 0 ) out vec4 FragColor;

float ggxDistribution( float nDotH ) {
  float alpha2 = Material.Rough * Material.Rough * Material.Rough * Material.Rough;
  float d = (nDotH * nDotH) * (alpha2 - 1) + 1;
  return alpha2 / (PI * d * d);
}

float geomSmith( float dotProd ) {
  float k = (Material.Rough + 1.0) * (Material.Rough + 1.0) / 8.0;
  float denom = dotProd * (1 - k) + k;
  return 1.0 / denom;
}

vec3 schlickFresnel( float lDotH ) {
  vec3 f0 = vec3(0.04);
  if( Material.Metal ) {
    f0 = Material.Color;
  }
  return f0 + (1 - f0) * pow(1.0 - lDotH, 5);
}
```

```
vec3 microfacetModel( int lightIdx, vec3 position, vec3 n ) {
  vec3 diffuseBrdf = vec3(0.0);  // Metallic
  if( !Material.Metal ) {
    diffuseBrdf = Material.Color;
  }

  vec3 l = vec3(0.0),
    lightI = Light[lightIdx].L;
  if( Light[lightIdx].Position.w == 0.0 ) { // Directional light
    l = normalize(Light[lightIdx].Position.xyz);
  } else {                                  // Positional light
    l = Light[lightIdx].Position.xyz - position;
    float dist = length(l);
    l = normalize(l);
    lightI /= (dist * dist);
  }

  vec3 v = normalize( -position );
  vec3 h = normalize( v + l );
  float nDotH = dot( n, h );
  float lDotH = dot( l, h );
  float nDotL = max( dot( n, l ), 0.0 );
  float nDotV = dot( n, v );
  vec3 specBrdf = 0.25 * ggxDistribution(nDotH) * schlickFresnel(lDotH) * geomSmith(nDotL) * geomSmith(nDotV);

  return (diffuseBrdf + PI * specBrdf) * lightI * nDotL;
}

void main() {
  vec3 sum = vec3(0);
  vec3 n = normalize(Normal);
  for( int i = 0; i < 3; i++ ) {
    sum += microfacetModel(i, Position, n);
  }

  // Gamma
  sum = pow( sum, vec3(1.0/2.2) );

  FragColor = vec4(sum, 1);
}
```

scenebasic_uniform.h:

```
private:
    GLSLProgram prog;

    Plane plane;
    std::unique_ptr<ObjMesh> mesh;
    Teapot teapot;

    float tPrev, lightAngle, lightRotationSpeed;
    glm::vec4 lightPos;

    void setMatrices();
    void compile();
    void drawScene();
    void drawFloor();
    void drawSpot(const glm::vec3& pos, float rough, int metal, const glm::vec3& color);
```

scenebasic_uniform.cpp:

For constructor and initScene() use this:

```
//constructor for torus
SceneBasic_Uniform::SceneBasic_Uniform() : plane(20, 20, 1, 1), teapot(5, glm::mat4(1.0f)),
                                           tPrev(0.0f), lightPos(5.0f, 5.0f, 5.0f, 1.0f)
{
    mesh = ObjMesh::load("../Project_Template/media/spot/spot_triangulated.obj");
}

void SceneBasic_Uniform::initScene()
{
    compile();

    glClearColor(0.1f, 0.1f, 0.2f, 1.0f);

    glEnable(GL_DEPTH_TEST);

    view = glm::lookAt(
        glm::vec3(0.0f, 4.0f, 7.0f),
        glm::vec3(0.0f, 0.0f, 0.0f),
        glm::vec3(0.0f, 1.0f, 0.0f)
    );

    projection = glm::perspective(glm::radians(50.0f), (float)width / height, 0.5f, 100.0f);

    lightAngle = 0.0f;
    lightRotationSpeed = 1.5f;

    prog.setUniform("Light[0].L", glm::vec3(45.0f));
    prog.setUniform("Light[0].Position", view * lightPos);
    prog.setUniform("Light[1].L", glm::vec3(0.3f));
    prog.setUniform("Light[1].Position", glm::vec4(0, 0.15f, -1.0f, 0));
    prog.setUniform("Light[2].L", glm::vec3(45.0f));
    prog.setUniform("Light[2].Position", view * glm::vec4(-7, 3, 7, 1));
}
```

For update(), render(), setMatrices() and resize() use this:

```cpp
void SceneBasic_Uniform::update( float t )
{
    float deltaT = t - tPrev;

    if (tPrev == 0.0f)
        deltaT = 0.0f;

    tPrev = t;

    if (animating())
    {
        lightAngle = glm::mod(lightAngle + deltaT * lightRotationSpeed, glm::two_pi<float>());
        lightPos.x = glm::cos(lightAngle) * 7.0f;
        lightPos.y = 3.0f;
        lightPos.z = glm::sin(lightAngle) * 7.0f;
    }
}

void SceneBasic_Uniform::render()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    prog.setUniform("Light[0].Position", view * lightPos);
    drawScene();
}

void SceneBasic_Uniform::setMatrices()
{
    glm::mat4 mv = view * model;
    prog.setUniform("ModelViewMatrix", mv);
    prog.setUniform("NormalMatrix", glm::mat3(mv));
    prog.setUniform("MVP", projection * mv);
}

void SceneBasic_Uniform::resize(int w, int h)
{
    //setup the ciewport and the projection matrix
    glViewport(0, 0, w, h);
    width = w;
    height = h;
    projection = glm::perspective(glm::radians(60.0f), (float)width / height, 0.3f, 100.0f);
}
```

For drawScene() and drawFloor() use this:

```
void SceneBasic_Uniform::drawScene() {
    drawFloor();

    // Draw dielectric cows with varying roughness
    int numCows = 9;
    glm::vec3 cowBaseColor(0.1f, 0.33f, 0.97f);
    for (int i = 0; i < numCows; i++) {
        float cowX = i * (10.0f / (numCows - 1)) - 5.0f;
        float rough = (i + 1) * (1.0f / numCows);
        drawSpot(glm::vec3(cowX, 0, 0), rough, 0, cowBaseColor);
    }

    // Draw metal cows
    float metalRough = 0.43f;
    // Gold
    drawSpot(glm::vec3(-3.0f, 0.0f, 3.0f), metalRough, 1, glm::vec3(1, 0.71f, 0.29f));
    // Copper
    drawSpot(glm::vec3(-1.5f, 0.0f, 3.0f), metalRough, 1, glm::vec3(0.95f, 0.64f, 0.54f));
    // Aluminum
    drawSpot(glm::vec3(-0.0f, 0.0f, 3.0f), metalRough, 1, glm::vec3(0.91f, 0.92f, 0.92f));
    // Titanium
    drawSpot(glm::vec3(1.5f, 0.0f, 3.0f), metalRough, 1, glm::vec3(0.542f, 0.497f, 0.449f));
    // Silver
    drawSpot(glm::vec3(3.0f, 0.0f, 3.0f), metalRough, 1, glm::vec3(0.95f, 0.93f, 0.88f));

}

void SceneBasic_Uniform::drawFloor() {
    model = glm::mat4(1.0f);
    prog.setUniform("Material.Rough", 0.9f);
    prog.setUniform("Material.Metal", 0);
    prog.setUniform("Material.Color", glm::vec3(0.2f));
    model = glm::translate(model, glm::vec3(0.0f, -0.75f, 0.0f));

    setMatrices();
    plane.render();
}
```

For drawSpot() use this:

```
void SceneBasic_Uniform::drawSpot(const glm::vec3& pos, float rough, int metal, const glm::vec3& color) {
    model = glm::mat4(1.0f);
    prog.setUniform("Material.Rough", rough);
    prog.setUniform("Material.Metal", metal);
    prog.setUniform("Material.Color", color);
    model = glm::translate(model, pos);
    model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));

    setMatrices();
    mesh->render();
}
```

That’s it.