計算機圖學 HW2 Report

1. 實作

1.1 TODO#0 更改視窗標題

把glfwSetWindowTitle()的第二個參數改成"HW2 - 311552013"。

```
glfwSetWindowTitle(window, "HW2 - 311552013");
```

1.2 TODO#1-1 註解掉範例model並拿掉要使用的model的註解

```
/*Model* m = new Model();
  float pos[] = {-1, 0, -1, -1, 0, 1, 1, 0, 1, 1, 0, -1};
  for (int i = 0; i < 12; i++) {
    m->positions.push_back(pos[i]);
  }
  m->numVertex = 4;
  m->drawMode = GL_QUADS;
  ctx.models.push_back(m);*/

Model* m = Model::fromObjectFile("../assets/models/cube/cube.obj");
m->textures.push_back(createTexture("../assets/models/cube/texture.bmp"));
m->modelMatrix = glm::scale(m->modelMatrix, glm::vec3(0.4f, 0.4f, 0.4f));
ctx.models.push_back(m);

m = Model::fromObjectFile("../assets/models/Mugs/Models/Mug_obj3.obj");
m->textures.push_back(createTexture("../assets/models/Mugs/Textures/Mug_C.png"));
m->textures.push_back(createTexture("../assets/models/Mugs/Textures/Mug_T.png"));
m->modelMatrix = glm::scale(m->modelMatrix, glm::vec3(6.0f, 6.0f, 6.0f));
ctx.models.push_back(m);
```

1.3 TODO#1-2 註解掉範例object並拿掉要使用的object的註解

```
/*ctx.objects.push_back(new Object(0, glm::translate(glm::identity<glm::mat4>(),
glm::vec3(0, 0 ,0))));*/

  ctx.objects.push_back(new Object(0, glm::translate(glm::identity<glm::mat4>(),
glm::vec3(1.5, 0.2, 2))));
  (*ctx.objects.rbegin())->material = mFlatwhite;
  ctx.objects.push_back(new Object(0, glm::translate(glm::identity<glm::mat4>(),
glm::vec3(2.5, 0.2, 2))));
  (*ctx.objects.rbegin())->material = mShinyred;
  ctx.objects.push_back(new Object(0, glm::translate(glm::identity<glm::mat4>(),
glm::vec3(3.5, 0.2, 2))));
  (*ctx.objects.rbegin())->material = mClearblue;
  ctx.objects.push_back(new Object(1, glm::translate(glm::identity<glm::mat4>(),
glm::vec3(3, 0.3, 3))));
  ctx.objects.push_back(new Object(1, glm::translate(glm::identity<glm::mat4>(),
glm::vec3(4, 0.3, 3))));
  (*ctx.objects.rbegin())->textureIndex = 1;
```

1.4 TODO#1 載入obj檔

先將頂點座標、材質座標、法向量分別存到v, vt, vn陣列裡，當遇到f時，先使用face_parser()函數得到該次v, vt, vn的index(vi, vti, vni)，並將相對應的資料真正存入class裡。

```
  std::string line = "";
  std::string prefix = "";
  std::stringstream ss;
  std::vector<float> v, vt, vn;
```

```
  while (getline(ObjFile, line)) {
    ss.clear();
    ss.str(line);
    ss >> prefix;
    if (prefix == "v") {
      float vx, vy, vz;
      ss >> vx >> vy >> vz;
      v.push_back(vx);
      v.push_back(vy);
      v.push_back(vz);
    } else if (prefix == "vt") {
      float vtx, vty;
      ss >> vtx >> vty;
      vt.push_back(vtx);
      vt.push_back(vty);
    } else if (prefix == "vn") {
      float vnx, vny, vnz;
      ss >> vnx >> vny >> vnz;
      vn.push_back(vnx);
      vn.push_back(vny);
      vn.push_back(vnz);
    } else if (prefix == "f") {
      for (int i = 0; i < 3; i++) {
        std::string p;
        ss >> p;
        int vi, vti, vni;
        face_parser(p, vi, vti, vni);
        m->positions.push_back(v[(vi - 1) * 3]);
        m->positions.push_back(v[(vi - 1) * 3 + 1]);
        m->positions.push_back(v[(vi - 1) * 3 + 2]);
        m->texcoords.push_back(vt[(vti - 1) * 2]);
        m->texcoords.push_back(vt[(vti - 1) * 2 + 1]);
        m->normals.push_back(vn[(vni - 1) * 3]);
        m->normals.push_back(vn[(vni - 1) * 3 + 1]);
        m->normals.push_back(vn[(vni - 1) * 3 + 2]);
        m->numVertex++;
      }
    }
  }
...
void face_parser(std::string str, int& v, int& vt, int& vn)
{
  v = 0;
  vt = 0;
  vn = 0;
  int i = 0;
  for (; i < str.size() && str[i]!='/'; i++) {
    v = v * 10 + (int)(str[i] - '0');
  }
  i++;
  for (; i < str.size() && str[i] != '/'; i++) {
    vt = vt * 10 + (int)(str[i] - '0');
  }
  i++;
```

```
  for (; i < str.size() && str[i] != '/'; i++) {
    vn = vn * 10 + (int)(str[i] - '0');
  }
}
```

1.5 TODO#2-1 basic shader

Basic shader較為單純，在vertex shader中，依照公式將projection, view, model等矩陣相乘，並將材質座標傳給fragment shader;在fragment shader中，因為有使用sample2D先將材質載入，使用texture()將材質座標傳遞給shader即可。

```
gl_Position = Projection * ViewMatrix * ModelMatrix * vec4(position, 1.0);
TexCoord = texCoord;
```

```
color = texture(ourTexture, TexCoord);
```

1.6 TODO#2-2 將資料傳到vertex buffer

我的作法是依序將一組一組的資料{vx, vy, vz, nx, ny, nz, tx, ty}放到combined陣列中，再呼叫3次傳值函數傳給shader中相對應layout的變數。

```
  int num_model = (int)ctx->models.size();
  VAO = new GLuint[num_model];
  glGenVertexArrays(num_model, VAO);
  for (int i = 0; i < num_model; i++) {
    // bind VAO
    glBindVertexArray(VAO[i]);

    // get model
    Model* model = ctx->models[i];

    //combine positions, normals, textures to one vector
    std::vector<float> combined;
    for (int j = 0; j < model->numVertex; j++) {
      combined.push_back(model->positions[j * 3]);
      combined.push_back(model->positions[j * 3 + 1]);
      combined.push_back(model->positions[j * 3 + 2]);
      combined.push_back(model->normals[j * 3]);
      combined.push_back(model->normals[j * 3 + 1]);
      combined.push_back(model->normals[j * 3 + 2]);
      combined.push_back(model->texcoords[j * 2]);
      combined.push_back(model->texcoords[j * 2 + 1]);
    }

    // generate and bind VBO
    GLuint VBO[1];
    glGenBuffers(1, VBO);

    // pass data to buffer
    glBindBuffer(GL_ARRAY_BUFFER, VBO[0]);
    glBufferData(GL_ARRAY_BUFFER, sizeof(float) * combined.size(), combined.data(),
```

```
GL_STATIC_DRAW);

    // set attributes
    // position
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)0);
    // normal
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)(3 *
sizeof(float)));
    // texture
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)(6 *
sizeof(float)));
  }
```

1.7 TODO#2-3 將資料傳到basic shader

這部分主要是處理shader中uniform變數和畫出頂點的部分。Basic shader中的uniform變數有:projection, view, model matrix以及貼圖材質，將其傳入shader中，最後再綁定材質並繪出頂點。

```
  glUseProgram(programId);
  int obj_num = (int)ctx->objects.size();
  for (int i = 0; i < obj_num; i++) {
    int modelIndex = ctx->objects[i]->modelIndex;
    glBindVertexArray(VAO[modelIndex]);

    Model* model = ctx->models[modelIndex];
    const float* p = ctx->camera->getProjectionMatrix();
    GLint pmatLoc = glGetUniformLocation(programId, "Projection");
    glUniformMatrix4fv(pmatLoc, 1, GL_FALSE, p);

    const float* v = ctx->camera->getViewMatrix();
    GLint vmatLoc = glGetUniformLocation(programId, "ViewMatrix");
    glUniformMatrix4fv(vmatLoc, 1, GL_FALSE, v);

    const float* m = glm::value_ptr(ctx->objects[i]->transformMatrix * model->modelMatrix);
    GLint mmatLoc = glGetUniformLocation(programId, "ModelMatrix");
    glUniformMatrix4fv(mmatLoc, 1, GL_FALSE, m);

    glUniform1i(glGetUniformLocation(programId, "ourTexture"), 0);
    glBindTexture(GL_TEXTURE_2D, model->textures[ctx->objects[i]->textureIndex]);
    glDrawArrays(model->drawMode, 0, model->numVertex);
  }
  glUseProgram(0);
```

1.8 TODO#3-1 新增木板

設定好頂點、法向量、材質座標，建立一個model object，加入model list即可。需要注意提示中所說的GL_REPEAT在材質座標超過0~1的範圍時，會自動用重複的方式填補，因此這邊材質座標的部分我設定為0~2之間。

```
std::vector<float> p = {
    -1.0, 0.0, -1.0,
    -1.0, 0.0, 1.0,
    1.0, 0.0, 1.0,
    1.0, 0.0, -1.0
  };
  std::vector<float> n = {
    0.0, 1.0, 0.0,
    0.0, 1.0, 0.0,
    0.0, 1.0, 0.0,
    0.0, 1.0, 0.0
  };
  std::vector<float> t = {
    0.0, 0.0,
    0.0, 2.0,
    2.0, 2.0,
    2.0, 0.0
  };

  m = new Model();
  m->positions.insert(m->positions.end(), p.begin(), p.end());
  m->normals.insert(m->normals.end(), n.begin(), n.end());
  m->texcoords.insert(m->texcoords.end(), t.begin(), t.end());
  m->textures.push_back(createTexture("../assets/models/Wood_maps/AT_Wood.jpg"));
  m->modelMatrix = glm::scale(m->modelMatrix, glm::vec3(4.096f, 1.0f, 2.56f));
  m->numVertex = 4;
  m->drawMode = GL_QUADS;
  ctx.models.push_back(m);
```

1.9 TODO#3-2 將木板加入畫面中

剛剛僅是設定木板的屬性，因此在這邊一個木板實體加入到object list中。

```
ctx.objects.push_back(new Object(2, glm::translate(glm::identity<glm::mat4>(),
glm::vec3(4.096, 0.0, 2.56))));
```

1.10 TODO#4-1 light shader

在vertex shader中，除了與basic shader一樣依照公式將projection, view, model等矩陣相乘之外，還需要計算position和mormal在world space的值並傳給fragment shader。在fragment shader中則是各光源的計算，每種光源都分成ambient, diffuse, specular三個部分。

- ambient: 直接給予一場景光源。
- diffuse: 計算法向量與光源的夾角，夾角愈小，給予愈強的光。
- specular: 計算光源反射方向與視野方向的夾角，夾角愈小，給予愈強的光。

在spot light中，若diffuse和specular超出範圍，則直接跳過不予計算。

```
  gl_Position = Projection * ViewMatrix * ModelMatrix *   vec4(position, 1.0);
  TexCoord = texCoord;
  FragPos = vec3(ModelMatrix * vec4(position, 1.0f));
  Normal = vec3(ModelNormalMatrix * vec4(normal, 1.0));
```

```
    vec4 dAmbient = vec4(0.0);
    vec4 dDiffuse = vec4(0.0);
    vec4 dSpecular = vec4(0.0);
    vec4 pAmbient = vec4(0.0);
    vec4 pDiffuse = vec4(0.0);
    vec4 pSpecular = vec4(0.0);
    vec4 sAmbient = vec4(0.0);
    vec4 sDiffuse = vec4(0.0);
    vec4 sSpecular = vec4(0.0);
    vec4 cAmbient = vec4(0.0);
    vec4 cDiffuse = vec4(0.0);
    vec4 cSpecular = vec4(0.0);

    if(dl.enable==1){ // directional light
        // ambient
        dAmbient = vec4(dl.lightColor*material.ambient, 1.0);

        //diffuse
        float ddCoef = dot(normalize((-1)*dl.direction), normalize(Normal));
        dDiffuse = vec4(dl.lightColor*material.diffuse, 1.0) * max(ddCoef, 0.0);

        // specular
        vec3 reflect_dir = reflect(dl.direction, normalize(Normal));
        vec3 view_dir = viewPos - FragPos;
        float dsCoef = dot(normalize(reflect_dir), normalize(view_dir));
        dSpecular = vec4(dl.lightColor*material.specular, 1.0) * pow(max(dsCoef, 0.0),
material.shininess);
    }
    if(pl.enable==1){ // point light
        // attenuation
        float dist = length(FragPos-pl.position);
        float attenuation = 1.0 / (pl.constant + pl.linear*dist + pl.quadratic*dist*dist);

        // ambient
        pAmbient = vec4(pl.lightColor*material.ambient, 1.0) * attenuation;

        // diffuse
        vec3 light_dir = FragPos - pl.position;
        float pdCoef = dot(normalize((-1)*light_dir), normalize(Normal));
        pDiffuse = vec4(pl.lightColor*material.diffuse, 1.0) * max(pdCoef, 0.0) *
attenuation;

        // specular
        vec3 reflect_dir = reflect(light_dir, normalize(Normal));
        vec3 view_dir = viewPos - FragPos;
        float psCoef = dot(normalize(reflect_dir), normalize(view_dir));
        pSpecular = vec4(pl.lightColor*material.specular, 1.0) * pow(max(psCoef, 0.0),
material.shininess) * attenuation;
    }
    if(sl.enable==1){
        // attenuation
        float dist = length(FragPos-sl.position);
        float attenuation = 1.0 / (sl.constant + sl.linear*dist + sl.quadratic*dist*dist);
```

```
        // ambient
        sAmbient = vec4(sl.lightColor*material.ambient, 1.0) * attenuation;

        // check if position is in the spotlight
        vec3 light_dir = FragPos - sl.position;
        if(dot(normalize(light_dir), normalize(sl.direction))>sl.cutOff){
            // diffuse
            float sdCoef = dot(normalize((-1)*light_dir), normalize(Normal));
            sDiffuse = vec4(sl.lightColor*material.diffuse, 1.0) * max(sdCoef, 0.0) *
attenuation;

            // specular
            vec3 reflect_dir = reflect(light_dir, normalize(Normal));
            vec3 view_dir = viewPos - FragPos;
            float ssCoef = dot(normalize(reflect_dir), normalize(view_dir));
            sSpecular = vec4(sl.lightColor*material.specular, 1.0) * pow(max(ssCoef, 0.0),
material.shininess) * attenuation;
        }
    }
    cAmbient = dAmbient + pAmbient + sAmbient;
    cDiffuse = dDiffuse + pDiffuse + sDiffuse;
    cSpecular = dSpecular + pSpecular + sSpecular;
    color = (cAmbient + cDiffuse + cSpecular) * texture(ourTexture, TexCoord);
```

1.11 TODO#4-2 將資料傳到vertex buffer

這裡所做的事情基本上與basic shader相同。

```
  int num_model = (int)ctx->models.size();
  VAO = new GLuint[num_model];

  glGenVertexArrays(num_model, VAO);
  for (int i = 0; i < num_model; i++) {
    // bind VAO
    glBindVertexArray(VAO[i]);

    // get model
    Model* model = ctx->models[i];

    // combine positions, normals, textures to one vector
    std::vector<float> combined;
    for (int j = 0; j < model->numVertex; j++) {
      combined.push_back(model->positions[j * 3]);
      combined.push_back(model->positions[j * 3 + 1]);
      combined.push_back(model->positions[j * 3 + 2]);
      combined.push_back(model->normals[j * 3]);
      combined.push_back(model->normals[j * 3 + 1]);
      combined.push_back(model->normals[j * 3 + 2]);
      combined.push_back(model->texcoords[j * 2]);
      combined.push_back(model->texcoords[j * 2 + 1]);
    }

    // generate and bind VBO
    GLuint VBO[1];
```

```
    glGenBuffers(1, VBO);

    // pass data to buffer
    glBindBuffer(GL_ARRAY_BUFFER, VBO[0]);
    glBufferData(GL_ARRAY_BUFFER, sizeof(float) * combined.size(), combined.data(),
GL_STATIC_DRAW);

    // set attributes
    // position
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)0);
    // normal
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)(3 *
sizeof(float)));
    // texture
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)(6 *
sizeof(float)));
  }
```

1.12 TODO#4-3 將資料傳到light shader

這裡所做的事情基本上與basic shader相同。

因為light shader的uniform參數實在太多了，每次要先取location再給值有點麻煩，因此決定先在program.h中先包一層函數來呼叫，簡化程式碼並增加可讀性。

```
  int obj_num = (int)ctx->objects.size();
  for (int i = 0; i < obj_num; i++) {
    int modelIndex = ctx->objects[i]->modelIndex;
    glBindVertexArray(VAO[modelIndex]);

    // pass variables to shader
    // camera
    Model* model = ctx->models[modelIndex];
    Camera* camera = ctx->camera;
    Object* object = ctx->objects[i];
    setMat4("Projection", camera->getProjectionMatrix());
    setMat4("ViewMatrix", camera->getViewMatrix());
    glm::mat4 modelMatrix = object->transformMatrix * model->modelMatrix;
    glm::mat4 modelNormalMatrix = glm::transpose(glm::inverse(modelMatrix));
    setMat4("ModelMatrix", glm::value_ptr(modelMatrix));
    setMat4("ModelNormalMatrix", glm::value_ptr(modelNormalMatrix));
    setVec3("viewPos", camera->getPosition());

    // texture
    setInt("ourTexture", 0);
    glBindTexture(GL_TEXTURE_2D, model->textures[ctx->objects[i]->textureIndex]);

    // material
    Material material = ctx->objects[i]->material;
    setVec3("material.ambient", glm::value_ptr(material.ambient));
    setVec3("material.diffuse", glm::value_ptr(material.diffuse));
    setVec3("material.specular", glm::value_ptr(material.specular));
```

```
    setFloat("material.shininess", material.shininess);

    // directional light
    setInt("dl.enable", ctx->directionLightEnable);
    setVec3("dl.direction", glm::value_ptr(ctx->directionLightDirection));
    setVec3("dl.lightColor", glm::value_ptr(ctx->directionLightColor));

    // point light
    setInt("pl.enable", ctx->pointLightEnable);
    setVec3("pl.position", glm::value_ptr(ctx->pointLightPosition));
    setVec3("pl.lightColor", glm::value_ptr(ctx->pointLightColor));
    setFloat("pl.constant", ctx->pointLightConstant);
    setFloat("pl.linear", ctx->pointLightLinear);
    setFloat("pl.quadratic", ctx->pointLightQuardratic);

    // spot light
    setInt("sl.enable", ctx->spotLightEnable);
    setVec3("sl.position", glm::value_ptr(ctx->spotLightPosition));
    setVec3("sl.direction", glm::value_ptr(ctx->spotLightDirection));
    setVec3("sl.lightColor", glm::value_ptr(ctx->spotLightColor));
    setFloat("sl.cutOff", ctx->spotLightCutOff);
    setFloat("sl.constant", ctx->spotLightConstant);
    setFloat("sl.linear", ctx->spotLightLinear);
    setFloat("sl.quadratic", ctx->spotLightQuardratic);

    glDrawArrays(model->drawMode, 0, model->numVertex);
  }
```

```
void setMat4(const char* varname, const float* data) {
    GLint loc = glGetUniformLocation(programId, varname);
    glUniformMatrix4fv(loc, 1, GL_FALSE, data);
  }
  void setVec3(const char* varname, const float* data) {
    GLint loc = glGetUniformLocation(programId, varname);
    glUniform3fv(loc, 1, data);
  }
  void setFloat(const char* varname, const float data) {
    GLint loc = glGetUniformLocation(programId, varname);
    glUniform1f(loc, data);
  }
  void setInt(const char* varname, const int data) {
    GLint loc = glGetUniformLocation(programId, varname);
    glUniform1i(loc, data);
  }
```