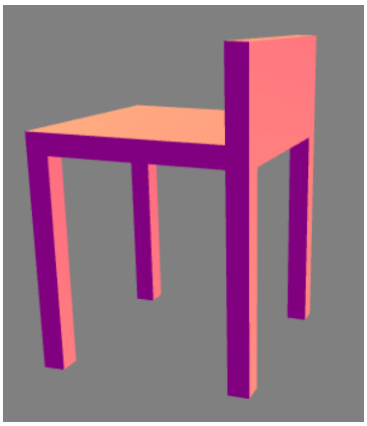
Interactive Graphics

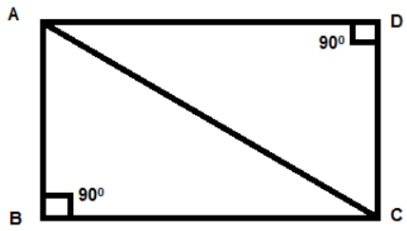
Small project - 1

1. Replace the cube with a simple model of a chair of 20 to 40 (maximum) vertices. Each vertex should have associated a normal (3 or 4 coordinates). Explain in the document how you chose the normal coordinates. Model the chair in its own coordinates.

The cube is replaced with a simple model of a chair, which consists of 40 vertices. The chair model which is shown below consists of 6 parts: 4 legs, seat and back. Each of the parts is a 3D cuboid (3D version of a rectangle) with different size measurements. Some parts of the chair (seat and back) share common vertices, 4 bottom vertices of the seat are the farthest vertices from each leg and 4 vertices of the back are from top vertices of 2 back-legs. So, putting them all together with corresponding coordinates we can get a simple chair model. The model of the chair in its own coordinates, i.e. at (0, 0, 0).



Each vertex is associated with a normal vector. The normal vectors are calculated and inserted to the array of the normals in *quad()* function, to which we give 4 vertices.



Lets say, A, B, C and D are the vertices of our quad then to calculate the normal at vertex A we should calculate the vectors along the two edges it is on, and then calculate the cross product of those vertices.

So, the normal vector at vertex A is CrossProduct((B - A), (C - A)).

We can repeat this for each vertex, although they will all be the same, because our quad is flat. If the surface is a flat polygon, the normal is the same at all points on the surface.

2. Choose an origin (different from the origin of the chair and include the rotation of the chair around the origin and along all three axes. Control with buttons/menus the axis of rotation, direction, speed, and start/stop.

A different origin is chosen as (0.5, 0.5, 0.5) and added a translation matrix in order to move the model to the new origin. As we know, the last transformation specified is first to be applied, so the translation matrix multiplies to *modelViewMatrix* last. The 3 rotation matrices along three axes (x, y, z) are specified and added. We can control the direction and speed of the rotation, also can start or stop the rotation by button, and change the rotation along x, y, z axes by buttons.

3. Add the viewer position (your choice), a perspective projection (your choice of parameters) and compute the ModelView and Projection matrices in the Javascript application. The viewer position and viewing volume should be controllable with buttons, sliders or menus. Please choose the initial parameters so that the object is clearly visible and the object is completely contained in the viewing volume. By changing the parameters you should be able to obtain situations where the object is partly or completely outside of the view volume.

For the viewer position, i.e. camera or eye position the *lookAt()* function from MV.js was applied, since it is the only one possible API for positioning the camera. *lookAt()* function consists of *eye, at* and *up* vectors: ***lookAt(eye, at, up)***. The *eye* vector consists of radius (distance of the camera from the origin) and angles *theta_view* and *phi* to move the camera. The *at* vector is the (x,y,z) coordinate of the point on the surface the viewer is looking at and *up* is (x,y,z) coordinate of some point in above.

The perspective projection function ***perspective(fovy, aspect, near, far)*** was applied. The angle fovy is the angle between the top and bottom planes of the clipping volume. The aspect ratio is width divided by height of the projection plane. The near plane is the location of it on the z-plane, which allows us to clip objects that are too close to the camera. The far plane is the location of it on the z-plane, which allows us to clip objects that are too distant from the camera.

4. Add a spotlight, and place it outside of the view volume. Assign to it all the necessary parameters (your choice). Add a button that turns the light on and off.

The spotlight with following parameters was applied:

```
var spotLightPosition = vec4(0.05, 0.1, 1.0, 1.0);
var spotLightAmbient = vec4(0.2, 0.2, 0.2, 1.0);
var spotLightDiffuse = vec4(1.0, 1.0, 1.0, 1.0);
```

```
var spotLightSpecular = vec4(1.0, 1.0, 1.0, 1.0);
var spotLightState = true; //for light is on/off
```

Ambient: The ambient light factor is generally used to moderate the effects of shadows from direct lighting, making shadows less dark than they otherwise would be.

Diffuse: Diffuse reflections are independent of the viewing direction, but depend on the direction of the light source with respect to the surface of the displayed object.

Specular: The higher the specular value, the brighter the resulting highlights.

A “Toggle Light” button is added that turns the light on and off.

5. Assign to the object a material with the relevant properties (your choice).

The material with properties Ambient, Diffuse, Specular was applied, since the material should match the terms in the light model.And additionally Shininess property added.

```
var materialAmbient = vec4(0.2, 0.2, 0.2, 1.0);

var materialDiffuse = vec4(1.0, 0.8, 0.0, 1.0);

var materialSpecular = vec4(1.0, 0.8, 0.0, 1.0);

var materialShininess = 100.0;
```

Shininess: The shininess coefficient describes the breadth of the angle of specular reflection.

6. Implement both per-vertex and per-fragment shading models.

Per-vertex and per-fragment shading models are implemented. The position and normalization with modelViewMatrix computings occur in the per-vertex model and the computed normalized spotlight location and uniform aColor is passed to the per-fragment model. The spotlight with corresponding properties (ambient, diffuse, specular and shininess) and color parameters are computed in the per-fragment model, also light switching occurs in per-fragment model.