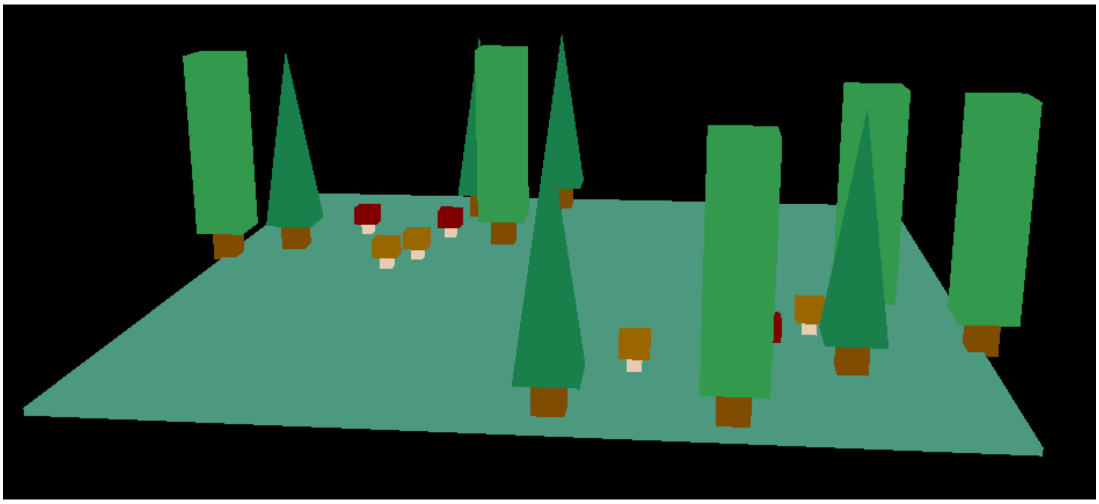

Example Forest Scene

We start inside a new folder with the name *ForestScene* or any equal name. In this folder we create a *ts*-file and an *html*-file. In the *html*-file we call our scripts like this:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Forest Scene</title>
    <script type="text/javascript" src="../../Core/Build/FudgeCore.js">
      </script> <!-- Path to FudgeCore -->
    <script type="text/javascript" src="ForestScene.js"></script>
</head>
<body>


</body>
</html>
```

Than we need to import Fudge, an eventListener and three Variables to our *ts*-file.

```typescript
///<reference types="../../Core/Build/FudgeCore.js"/> //Path to FudgeCore
namespace ExampleSceneForest {
    import ƒ = FudgeCore;
    window.addEventListener("DOMContentLoaded", init);
    let node: ƒ.Node;
    let camera: ƒ.Node;
    let viewPort: ƒ.Viewport;
}
```

We need also two functions: *init()* and *createMiniForest()*. Both have no return-value. The *init()*-function is needed for calling other functions and showing the final image. For this *init()* needs the three callings *ƒ.RenderManager.initialize()*, *createMiniForest()* and *viewPort.draw()*. With the calling *viewport.showSceneGraph()* can we see a scene graph in the web console of the browser.

```
function init(): void {
      ƒ.RenderManager.initialize();
      createMiniForest();
      viewPort.draw();
      viewPort.showSceneGraph();
}
```

createMiniForest() is the main function for creating our forest. It stores the variables for different nodes, colours and creates all the components of the forest.

We create now an empty node *forest* and all the colours we will need for the forest. Also we create a node *ground*, save the mesh component in the variable *cmpGroundMesh* and scale the ground to the preferred size.

Finally we add the ground to the topmost node, add a camera to the scene and create a viewport.

```
function createMiniForest(): void {
      let forest: ƒ.Node = new ƒ.Node("Forest");

      let clrLeaves: ƒ.Color = new ƒ.Color(0.2, 0.6, 0.3, 1);
      let clrNeedles: ƒ.Color = new ƒ.Color(0.1, 0.5, 0.3, 1);
      let clrTrunkTree: ƒ.Color = new ƒ.Color(0.5, 0.3, 0, 1);
      let clrCapMushroomBrown: ƒ.Color = new ƒ.Color(0.6, 0.4, 0, 1);
      let clrCapMushroomRed: ƒ.Color = new ƒ.Color(0.5, 0, 0, 1);
      let clrTrunkMushroom: ƒ.Color = new ƒ.Color(0.9, 0.8, 0.7, 1);
      let clrGround: ƒ.Color = new ƒ.Color(0.3, 0.6, 0.5, 1);

      let ground: ƒ.Node = createCompleteMeshNode("Ground",
      new ƒ.Material("Ground", ƒ.ShaderUniColor, new ƒ.CoatColored(clrGround))
            ,new ƒ.MeshCube());
      let cmpGroundMesh: ƒ.ComponentMesh = ground.getComponent
            (ƒ.ComponentMesh);

      cmpGroundMesh.pivot.scale(new ƒ.Vector3(6, 0.05, 6));

      node = ground;
      createViewport();
}
```

For the ground, we don't want an empty node. It will get a name, a material and a mesh. For this we need to write the function *createCompleteMeshNode()*.

```
function createCompleteMeshNode(_name: string, _material: ƒ.Material, _mesh: ƒ
            .Mesh): ƒ.Node {
        let node: ƒ.Node = new ƒ.Node(_name);

        let cmpMesh: ƒ.ComponentMesh = new ƒ.ComponentMesh(_mesh);
        let cmpMaterial: ƒ.ComponentMaterial = new ƒ.ComponentMaterial
            (_material);

        let cmpTransform: ƒ.ComponentTransform = new ƒ.ComponentTransform();
        node.addComponent(cmpMesh);
        node.addComponent(cmpMaterial);
        node.addComponent(cmpTransform);

        return node;
    }
```

To create the viewport we write the function *createViewport().*

It creates a canvas if there's not already one and initializes after this the viewport. It also generates a camera with the help of the function *createCamera()*.

```
function createViewport(_canvas: HTMLCanvasElement = null): void {
        if (!_canvas) {
            _canvas = document.createElement("canvas");
            _canvas.width = 800;
            _canvas.height = 600;
            document.body.appendChild(_canvas);
        }
        viewPort = new ƒ.Viewport();
        camera = createCamera();
        viewPort.initialize("viewport", node, camera.getComponent
            (ƒ.ComponentCamera), _canvas);
    }
```

```
function createCamera(_translation: ƒ.Vector3 = new ƒ.Vector3(1, 1, 10), _look
At: ƒ.Vector3 = new ƒ.Vector3()): ƒ.Node {
        let camera: ƒ.Node = new ƒ.Node("Camera");
        let cmpTransform: ƒ.ComponentTransform = new ƒ.ComponentTransform();
        cmpTransform.local.translate(_translation);
        cmpTransform.local.lookAt(_lookAt);
        camera.addComponent(cmpTransform);
        let cmpCamera: ƒ.ComponentCamera = new ƒ.ComponentCamera();
        cmpCamera.projectCentral(1, 45, ƒ.FIELD_OF_VIEW.DIAGONAL);
        camera.addComponent(cmpCamera);
        return camera;
    }
```

Now we can start to program the components of the forest: Broadleaf trees, conifers and mushrooms.

The function *createBroadleaf()* receives a string for the name, a colour for the trunk, a colour for the leaves as well as the position and the scale of the tree.
Then we add an empty node, which receives the value of the string *_name* and command the function to return the value of this node.

```
function createBroadleaf(_name: string, _clrTrunk: ƒ.Color, _clrTop: ƒ.Color,
      _pos: ƒ.Vector3, _scale: ƒ.Vector3): ƒ.Node {

      let tree: ƒ.Node = new ƒ.Node(_name);

```

```
    return tree;
}
```

As the next step, we add a second node to the function for the tree trunk. For this we use the previously defined value of *_clrTrunk*.

```
function createBroadleaf(_name: string, _clrTrunk: ƒ.Color, _clrTop: ƒ.Color,
    _pos: ƒ.Vector3, _scale: ƒ.Vector3): ƒ.Node {

    let tree: ƒ.Node = new ƒ.Node(_name);

    let treeTrunk: ƒ.Node = createCompleteMeshNode("TreeTrunk", new
        ƒ.Material("TrunkTree", ƒ.ShaderUniColor, new
        ƒ.CoatColored(_clrTrunk)), new ƒ.MeshCube);

    return tree;
}
```

After this we safe the mesh component of the node in a variable *cmpTrunkMesh*. With the help of this variable we change the scale and the position of the trunk so that it is placed on the ground.

```
function createBroadleaf(_name: string, _clrTrunk: ƒ.Color, _clrTop: ƒ.Color,
    _pos: ƒ.Vector3, _scale: ƒ.Vector3): ƒ.Node {

    let tree: ƒ.Node = new ƒ.Node(_name);

    let treeTrunk: ƒ.Node = createCompleteMeshNode("TreeTrunk", new
        ƒ.Material("TrunkTree", ƒ.ShaderUniColor, new
        ƒ.CoatColored(_clrTrunk)), new ƒ.MeshCube);

    let cmpTrunkMesh: ƒ.ComponentMesh =
        treeTrunk.getComponent(ƒ.ComponentMesh);

    cmpTrunkMesh.pivot.scale(_scale);
    cmpTrunkMesh.pivot.translateY(_scale.y / 2);

    return tree;
}
```

Next, we do the same to create the leaves of the tree but we give them the colour *_clrTop*.

```
function createBroadleaf(_name: string, _clrTrunk: ƒ.Color, _clrTop: ƒ.Color,
      _pos: ƒ.Vector3, _scale: ƒ.Vector3): ƒ.Node {

      let tree: ƒ.Node = new ƒ.Node(_name);

      let treeTrunk: ƒ.Node = createCompleteMeshNode("TreeTrunk", new
            ƒ.Material("TrunkTree", ƒ.ShaderUniColor, new
            ƒ.CoatColored(_clrTrunk)), new ƒ.MeshCube);

      let cmpTrunkMesh: ƒ.ComponentMesh =
            treeTrunk.getComponent(ƒ.ComponentMesh);

      cmpTrunkMesh.pivot.scale(_scale);
      cmpTrunkMesh.pivot.translateY(_scale.y / 2);

      let treeTop: ƒ.Node = createCompleteMeshNode("TreeTop", new
            ƒ.Material("TreeTop", ƒ.ShaderUniColor, new
            ƒ.CoatColored(_clrTop)), new ƒ.MeshCube);

      let cmpTreeTopMesh: ƒ.ComponentMesh =
            treeTop.getComponent(ƒ.ComponentMesh);

      cmpTreeTopMesh.pivot.scale(new ƒ.Vector3((_scale.x * 2), (_scale.y * 3),
            (_scale.z * 2)));

      cmpTreeTopMesh.pivot.translateY((_scale.y * 2));

      return tree;
}
```

Finally we add the leaves and the trunk to the parent node *tree* and give this node a transform component. With this, we can place the tree in the scene wherever we want.

```
function createBroadleaf(_name: string, _clrTrunk: ƒ.Color, _clrTop: ƒ.Color,
       _pos: ƒ.Vector3, _scale: ƒ.Vector3): ƒ.Node {

      let tree: ƒ.Node = new ƒ.Node(_name);

      let treeTrunk: ƒ.Node = createCompleteMeshNode("TreeTrunk", new
            ƒ.Material("TrunkTree", ƒ.ShaderUniColor, new
            ƒ.CoatColored(_clrTrunk)), new ƒ.MeshCube);

      let cmpTrunkMesh: ƒ.ComponentMesh =
            treeTrunk.getComponent(ƒ.ComponentMesh);

      cmpTrunkMesh.pivot.scale(_scale);
      cmpTrunkMesh.pivot.translateY(_scale.y / 2);

      let treeTop: ƒ.Node = createCompleteMeshNode("TreeTop", new
            ƒ.Material("TreeTop", ƒ.ShaderUniColor, new
            ƒ.CoatColored(_clrTop)), new ƒ.MeshCube);

      let cmpTreeTopMesh: ƒ.ComponentMesh =
            treeTop.getComponent(ƒ.ComponentMesh);

      cmpTreeTopMesh.pivot.scale(new ƒ.Vector3((_scale.x * 2), (_scale.y * 3),
            (_scale.z * 2)));

      cmpTreeTopMesh.pivot.translateY((_scale.y * 2));
      tree.appendChild(treeTop);
      tree.appendChild(treeTrunk);
      tree.addComponent(new ƒ.ComponentTransform);
      tree.cmpTransform.local.translate(_pos);

      return tree;
}
```

At this point we just need to call *createBroadleaf()* in *createMiniForest()*, build the code and start the *html*-file in the browser to see our tree.

```
function createMiniForest(): void {
        let forest: ƒ.Node = new ƒ.Node("Forest");

        let clrLeaves: ƒ.Color = new ƒ.Color(0.2, 0.6, 0.3, 1);
        let clrNeedles: ƒ.Color = new ƒ.Color(0.1, 0.5, 0.3, 1);
        let clrTrunkTree: ƒ.Color = new ƒ.Color(0.5, 0.3, 0, 1);
        let clrCapMushroomBrown: ƒ.Color = new ƒ.Color(0.6, 0.4, 0, 1);
        let clrCapMushroomRed: ƒ.Color = new ƒ.Color(0.5, 0, 0, 1);
        let clrTrunkMushroom: ƒ.Color = new ƒ.Color(0.9, 0.8, 0.7, 1);
        let clrGround: ƒ.Color = new ƒ.Color(0.3, 0.6, 0.5, 1);

        let ground: ƒ.Node = createCompleteMeshNode("Ground",
        new ƒ.Material("Ground", ƒ.ShaderUniColor, new ƒ.CoatColored(clrGround))
                ,new ƒ.MeshCube());
        let cmpGroundMesh: ƒ.ComponentMesh = ground.
                getComponent(ƒ.ComponentMesh);

        cmpGroundMesh.pivot.scale(new ƒ.Vector3(6, 0.05, 6));

        node = ground;
        createViewport();

        let broadleaf: ƒ.Node = createBroadleaf("BroadLeaf", clrTrunkTree,
                clrLeaves, new ƒ.Vector3(0, 0, 0), new ƒ.Vector3(0.2, 0.5, 0.2));

        node.appendChild(broadleaf);
}
```

To see our tree a little from the top we can save the transform component of the camera in a variable after we created the viewport and change the position of the camera.

```
function createMiniForest(): void {
        let forest: ƒ.Node = new ƒ.Node("Forest");

        let clrLeaves: ƒ.Color = new ƒ.Color(0.2, 0.6, 0.3, 1);
        let clrNeedles: ƒ.Color = new ƒ.Color(0.1, 0.5, 0.3, 1);
```

```
      let clrTrunkTree: ƒ.Color = new ƒ.Color(0.5, 0.3, 0, 1);
      let clrCapMushroomBrown: ƒ.Color = new ƒ.Color(0.6, 0.4, 0, 1);
      let clrCapMushroomRed: ƒ.Color = new ƒ.Color(0.5, 0, 0, 1);
      let clrTrunkMushroom: ƒ.Color = new ƒ.Color(0.9, 0.8, 0.7, 1);
      let clrGround: ƒ.Color = new ƒ.Color(0.3, 0.6, 0.5, 1);

      let ground: ƒ.Node = createCompleteMeshNode("Ground",
      new ƒ.Material("Ground", ƒ.ShaderUniColor, new ƒ.CoatColored(clrGround))
            ,new ƒ.MeshCube());
      let cmpGroundMesh: ƒ.ComponentMesh = ground.
            getComponent(ƒ.ComponentMesh);

      cmpGroundMesh.pivot.scale(new ƒ.Vector3(6, 0.05, 6));

      node = ground;
      createViewport();

      let cmpCamera: ƒ.ComponentTransform = camera.getComponent
            (ƒ.ComponentTransform);
        cmpCamera.local.translateY(2);
        cmpCamera.local.rotateX(-10);

      let broadleaf: ƒ.Node = createBroadleaf("BroadLeaf", clrTrunkTree,
            clrLeaves, new ƒ.Vector3(0, 0, 0), new ƒ.Vector3(0.2, 0.5, 0.2));

      node.appendChild(broadleaf);
}
```

Now we do the same as we have done with the broadleaf tree to create conifers and mushrooms.

```
function  createConifer(_name:  string,  _clrTrunk:  ƒ.Color,  _clrTop:  ƒ.Color,
      _pos: ƒ.Vector3, _scale: ƒ.Vector3): ƒ.Node {

      let tree: ƒ.Node = new ƒ.Node(_name);


      let treeTrunk: ƒ.Node = createCompleteMeshNode("TreeTrunk", new
            ƒ.Material("TrunkTree", ƒ.ShaderUniColor, new
            ƒ.CoatColored(_clrTrunk)), new ƒ.MeshCube);
```

```
    let cmpTrunkMesh: ƒ.ComponentMesh =
        treeTrunk.getComponent(ƒ.ComponentMesh);

    cmpTrunkMesh.pivot.scale(_scale);
    cmpTrunkMesh.pivot.translateY(_scale.y / 2);

    let treeTop: ƒ.Node = createCompleteMeshNode("TreeTop", new
        ƒ.Material("TreeTop", ƒ.ShaderUniColor, new
        ƒ.CoatColored(_clrTop)), new ƒ.MeshPyramid);

    let cmpTreeTopMesh: ƒ.ComponentMesh =
        treeTop.getComponent(ƒ.ComponentMesh);

    cmpTreeTopMesh.pivot.scale(new ƒ.Vector3((_scale.x * 2), (_scale.y * 3),
        (_scale.z * 2)));

    cmpTreeTopMesh.pivot.translateY((_scale.y / 2));

    tree.appendChild(treeTop);
    tree.appendChild(treeTrunk);

    tree.addComponent(new ƒ.ComponentTransform);
    tree.cmpTransform.local.translate(_pos);

    return tree;
}
```

```
function createMushroom(_name: string, _clrTrunk: ƒ.Color, _clrCap: ƒ.Color,
    _pos: ƒ.Vector3, _scale: ƒ.Vector3): ƒ.Node {

    let mushroom: ƒ.Node = new ƒ.Node(_name);
    let mushroomTrunk: ƒ.Node =
    createCompleteMeshNode("MushroomTrunk", new
        ƒ.Material("MushroomTrunk", ƒ.ShaderUniColor, new
        ƒ.CoatColored(_clrTrunk)), new ƒ.MeshCube);

    let cmpMesh: ƒ.ComponentMesh =
        mushroomTrunk.getComponent(ƒ.ComponentMesh);

    cmpMesh.pivot.scale(_scale);
    cmpMesh.pivot.translateY(_scale.y / 2);

    let mushroomCap: ƒ.Node =
        createCompleteMeshNode("MushroomCapRed", new
        ƒ.Material("MushroomCapRed", ƒ.ShaderUniColor, new
        ƒ.CoatColored(_clrCap)), new ƒ.MeshCube);

    let cmpCapMesh: ƒ.ComponentMesh =
        mushroomCap.getComponent(ƒ.ComponentMesh);

    cmpCapMesh.pivot.scale(new ƒ.Vector3((_scale.x * 2), (_scale.y - 0.05),
        (_scale.z * 2)));

    cmpCapMesh.pivot.translateY((_scale.y));

    mushroom.appendChild(mushroomCap);
    mushroom.appendChild(mushroomTrunk);

    mushroom.addComponent(new ƒ.ComponentTransform);
    mushroom.cmpTransform.local.translate(_pos);

    return mushroom;
}
```

In the final step we call *createBroadleaf()*, *createConifer()* and *createMushroom()* inside for-loops to create a lot of trees and mushrooms and add them to the root node.

```
function createMiniForest(): void {
    let forest: ƒ.Node = new ƒ.Node("Forest");

    let clrLeaves: ƒ.Color = new ƒ.Color(0.2, 0.6, 0.3, 1);
    let clrNeedles: ƒ.Color = new ƒ.Color(0.1, 0.5, 0.3, 1);
    let clrTrunkTree: ƒ.Color = new ƒ.Color(0.5, 0.3, 0, 1);
    let clrCapMushroomBrown: ƒ.Color = new ƒ.Color(0.6, 0.4, 0, 1);
    let clrCapMushroomRed: ƒ.Color = new ƒ.Color(0.5, 0, 0, 1);
    let clrTrunkMushroom: ƒ.Color = new ƒ.Color(0.9, 0.8, 0.7, 1);
    let clrGround: ƒ.Color = new ƒ.Color(0.3, 0.6, 0.5, 1);

    let ground: ƒ.Node = createCompleteMeshNode("Ground",
          new ƒ.Material("Ground", ƒ.ShaderUniColor, new
          ƒ.CoatColored(clrGround)), new ƒ.MeshCube());
    let cmpGroundMesh: ƒ.ComponentMesh = ground.getComponent
          (ƒ.ComponentMesh);

    cmpGroundMesh.pivot.scale(new ƒ.Vector3(6, 0.05, 6));
    node = ground;
    createViewport();

    let cmpCamera: ƒ.ComponentTransform = camera.getComponent
          (ƒ.ComponentTransform);
      cmpCamera.local.translateY(2);
      cmpCamera.local.rotateX(-10);

    //Creates a forest of broadleaves
      for (let i: number = 1; i <= 5; i++) {
          let plusOrMinus = Math.random() < 0.5 ? -1 : 1;
          let broadleaf: ƒ.Node = createBroadleaf("BroadLeaf" + i,
                clrTrunkTree, clrLeaves, new ƒ.Vector3(Math.random() * 4 *
                plusOrMinus, 0, Math.random() * 4 * plusOrMinus),
                new ƒ.Vector3(0.2, 0.5, 0.2));

          forest.appendChild(broadleaf);
      }


      //Creates a forest of conifers
      for (let i: number = 1; i <= 5; i++) {
          let plusOrMinus = Math.random() < 0.5 ? -1 : 1;
          let conifer: ƒ.Node = createConifer("Conifer" + i,
                clrTrunkTree, clrNeedles, new ƒ.Vector3(Math.random() * 3
                  * plusOrMinus, 0, Math.random() * 3 * plusOrMinus),
                  new ƒ.Vector3(0.2, 0.5, 0.2));

            forest.appendChild(conifer);
        }

        //Creates mushrooms
        for (let i: number = 1; i <= 4; i++) {
            let plusOrMinus = Math.random() < 0.5 ? -1 : 1;
            let mushroomRed: ƒ.Node = createMushroom("MushroomRed" + i,
                  clrTrunkMushroom, clrCapMushroomRed, new
                  ƒ.Vector3(Math.random() * 2 * plusOrMinus, 0,
                  Math.random() * 2 * plusOrMinus),
                  new ƒ.Vector3(0.1, 0.2, 0.1));

            let mushroomBrown: ƒ.Node = createMushroom("MushroomBrown" + i,
                  clrTrunkMushroom, clrCapMushroomBrown, new
                  ƒ.Vector3(Math.random() * 2 * plusOrMinus, 0,
                  Math.random() * 2 * plusOrMinus), new
                  ƒ.Vector3(0.1, 0.2, 0.1));

            forest.appendChild(mushroomRed);
            forest.appendChild(mushroomBrown);
        }

        node.appendChild(forest);
}
```