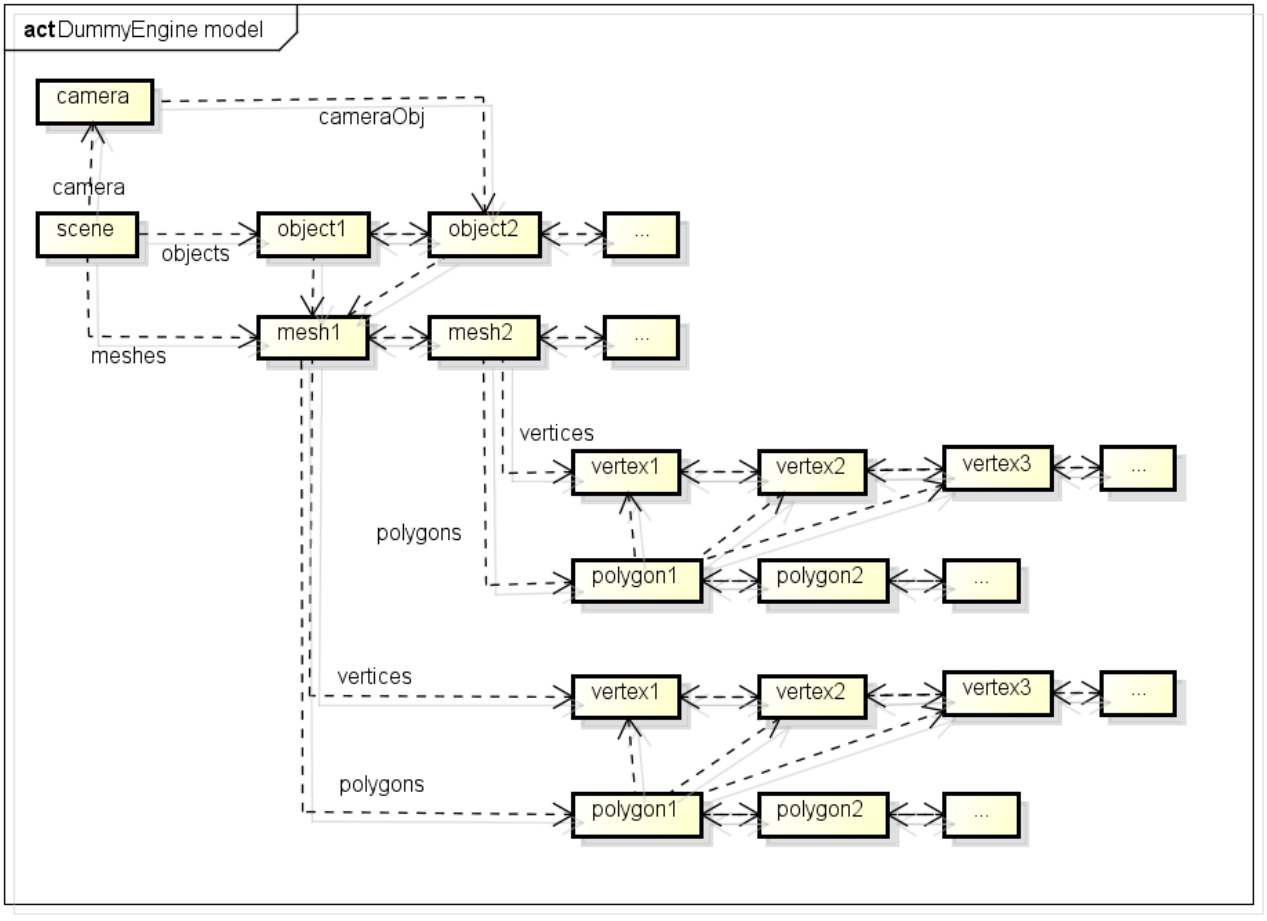
TOTEUTUSDOKUMENTTI

DummyEnginen toteutus oli varsin monimutkainen johtuen siitä, että olen kirjoittanut kaiken toiminnallisuuden itse. Keskitynkin tässä dokumentissa vain 3D-tilan tietorakenteiden selittämiseen ja jätän muunmuassa matriisiluokan käsittelyn kokonaan pois.

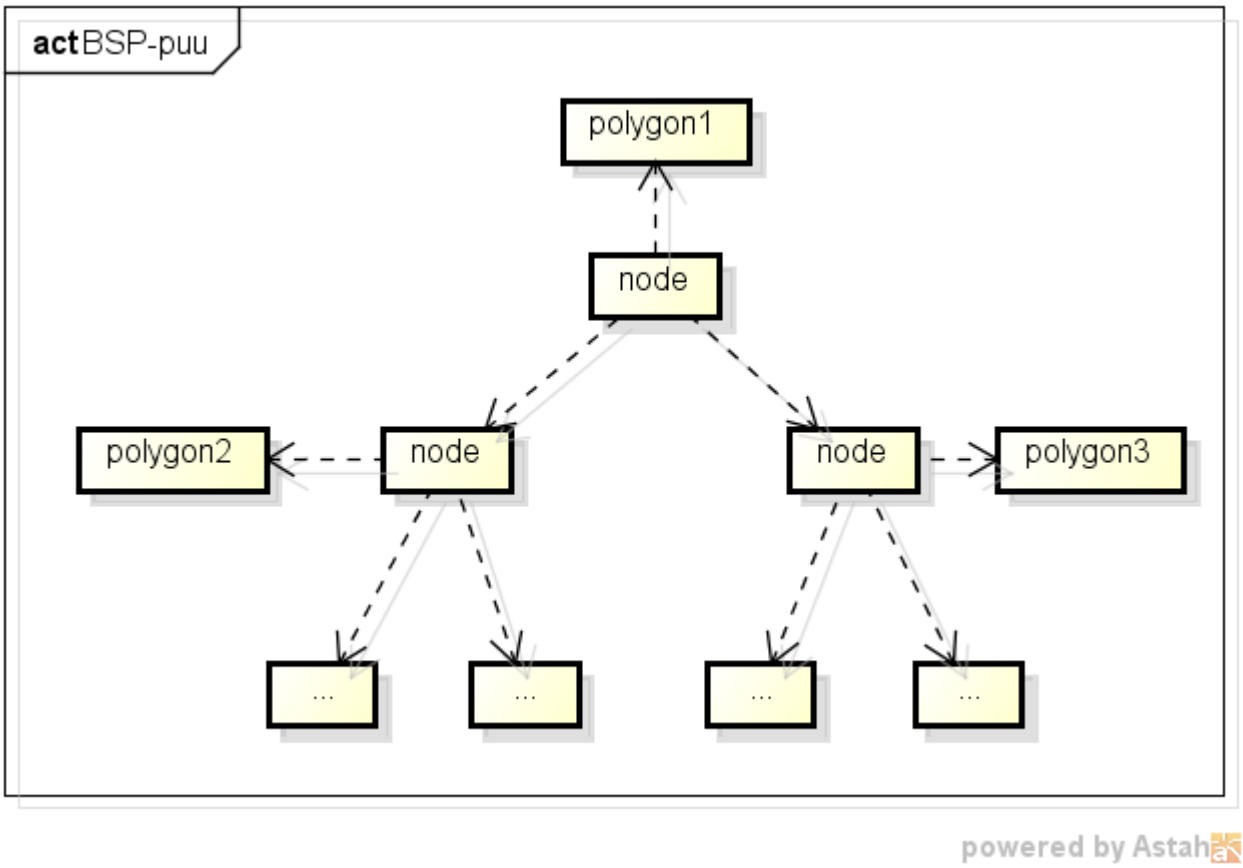
Ohjelman yleisrakenne on melko yksinkertainen. 3D-avaruuden pistettä kuvaa vertex-struktuuri, joka sisältää x-, y- ja z-koordinaattien lisäksi osoittimet verteksin eri koordinaattimatriiseihin. Vertekseille lasketaan piirtämisen eri vaiheessa viidet eri koordinaatit, joista tallennetaan neljä. Tasoa avaruudessa kuvaa polygon-struktuuri, joka sisältää osoittimet kolmeen verteksiin. Polygonille tallennetaan myös sen normaali. 3D-mallia kuvaa mesh-struktuuri. Se sisältää osoittimet linkitettyihin listoihin, jotka sisältävät mallin verteksit ja polygonit. Objektia 3D-avaruudessa kuvaa object-struktuuri. Objektit sisältävät osoittimen omiin koordinaatteihinsa, orientaatioonsa ja skaalaansa sekä osoittimen johonkin malliin, joka kertoo, mikä malli kyseisessä kohdassa on.

Tila-luokka hallinnoi kaikkea. Se sisältää osoittimet objektien ja mallien linkitettyihin listoihin. Siltä löytyy myös osoitin camera-struktuuriin, joka pitää huolta kuvakulmaan liittyvistä asioista. Camera-struktuuri sisältää myös osoittimen objektiin, joka toimii kameran silmäpisteenä (eye point).



Piirtämisen aputietorakenteena käytän BSP-puuta. Se luokittelee tilan polygonit sen mukaan, ovatko ne toistensa edessä vai takana ja asettaa ne binääripuun alkioiksi. Koska sama polygoni voi olla useamman objektin mallina ja siten esiintyä tilassa moneen kertaan eri paikoissa, tallennetaan BSP-puun alkioon myös

osoitin jokaisen polygonin ikkunakoordinaattien laskemiseksi. Tämän jälkeen ne voi piirtää mistä tahansa kuvakulmasta sisäjärjestyksessä.



O-ANALYYSI AIKA- JA TILAVAATIVUUDESTA

Piirtämisen aika- ja tilavaativuuteen vaikuttavat todella monet asiat aina pikselitasolta objektien määrään asti. Pelkästään algoritmien määrän takia rajoitun tarkastelemaan vain päätietorakenteen eli BSP-puun aika- ja tilavaativuutta.

BSP-puun noodin luova funktio on seuraavanlainen:

```
createBSPNode(polygon, matrixTransform)

        node = newNode()
        node->polygon = polygon
        node->matrixTransform = matrixTransform

        node->next = NULL
        node->prev = NULL

        return node
```

Koska kaikki käskyt ovat vakioaikaisia, on sen aikavaativuus O(1). Tilavaativuus on myös O(1), sillä puun alkio on vakiokokoinen.

BSP-puun luova algoritmi toimii siten, että se aluksi asettaa kaikki polygonit juuren front-listaan ja sen jälkeen käy ne läpi siirtäen alkioita tarvittaessa behind listaan. Tämän jälkeen front- ja behind-puun juuret käsitellään rekursiivisesti samalla tavalla. BSP-puun luova algoritmi toimii seuraavasti:

```
createBSPTree(scene)

        camMatrix = calculateCameraMatrix(scene)

        root = NULL
        prevNode = NULL

        for each obj in scene->objects

                world = matrixMultiply(obj->worldTransform, obj->scaleTransform)
                fullTransform = matrixMultiply(camMatrix, world)

                for each P in obj->mesh->polygons

                        calculateWorldCoordinates(P, world)

                        if( !doBackfaceCulling(scene->camera, P))

                                node = createBSPNode(P, fullTransform)

                                if( root == NULL)
                                        root = node
                                if( prevNode != NULL)
                                        prevNode->front = node
                                prevNode = node

        resolveBSPTree(root)

        return root
```

Funktiot calculateCameraMatrix, calculateWorldCoordinates ja doBackfaceCulling sisältävät vain tietyt laskutoimenpiteet, joten ne ovat vakioaikaisia. Myös createBSPNode on vakioaikainen. Funktion aikavaativuutta hallitsee siis objektien lukumäärä tilassa ja niiden polygonien määrä sekä rekursiivinen funktio resolveBSPTree(root). Funktion tilavaativuus on luokkaa O(P), missä P on tilan polygonien lukumäärä, sillä jokaiselle polygonille luodaan oma alkio BSP-puuhun.

Funktio resolveBSPTree on rekursiivinen kutsu, joka asettaa alkiot oikeaan järjestykseen puussa. Sen pseudokoodi on seuraavan näköinen:

```
resolveBSPTree(root)

        if( root == NULL)
                return

        for each node in root->front

                if( isInFrontOfPolygon(node->polygon, root->polygon)

                        tmp= node->front

                        addNodeBehind(root, node)

                        node = tmp

        resolveBSPTree(root->behind)
        resolveBSPTree(root->front)
```

Funktio isInFrontOfPolygon on vakioaikainen, sillä se tekee vain äärellisen määrän laskutoimituksia. Myöskin addNodeBehind on vakioaikainen, joten aikavaativuuden ratkaisee itse BSP-puun muoto. Ensimmäinen kutsu käy läpi kaikki polygonit, seuraavat kutsut vain osan polygoneista. Jokaisella tasolla tulee kuitenkin vertailtua n-1 polygonia, jos n on tason korkeus. Tällöin kokonaisuudessaan tulee tehtyä yhteensä n*(n+1)/2 vertailua, joten funktion aikavaativuus on luokkaa O(n^2), missä n on polygonien määrä. Funktion tilavaativuus on luokkaa O(P), missä P on tilan polygonien lukumäärä. Tämä johtuu siitä, että rekursiopinon koko saattaa pahimmillaan olla kaikkien polygonien lukumäärä, jos kaikki polygonit ovat peräkkäin.

Tästä johtuen createBSPTree:n aikavaativuus on luokkaa O(O*P + P^2), missä O on tilan objektien lukumäärä ja P polygonien lukumäärä. Funktion tilavaativuus on luokkaa O(P), sillä jokaiselle polygonille luodaan oma alkionsa puuhun ja rekursiopinon korkeus on pahimmillaan P. BSP-puun rakentaminen on aikaa vievä operaatio, mutta sen jälkeen piirtämisen pitäisi olla nopeaa. Käsitellään seuraavaksi traveBSPTree-funktio, jonka pseudokoodi on seuraavan näköinen:

```
travelBSPTree(root)

        if( root == NULL)
                return

        if( cameraInFrontOfPolygon(root->polygon))
                travelBSPTree(root->behind)
                drawPolygonSolid(root->polygon)
                travelBSPTree(root->front)
        else
                travelBSPTree(root->front)
                drawPolygonSolid(root->polygon)
                travelBSPTree(root->front)
```

Koska funktiot cameraInFrontOfPolygon ja drawPolygonSolid ovat vakioaikaisia, riippuu aikavaativuus vain polygonien määrästä P. Tilan, jossa on P polygonia, piirtäminen kestää siis lineaarisen ajan eli on luokkaa O(P) riippumatta kuvakulmasta. Tilavaativuus on luokkaa O(1).

PUUTTEITA & KEHITETTÄVÄÄ

BSP-puun toteutus ei ole tässä vielä täydellinen. Funktio vaatisi vielä metodin, joka paloittelisi toisensa leikkaavat polygonit osiin, jotka eivät enää leikkaa toisiaan. Leikkauspisteiden laskeminen on kuitenkin melko hankalaa, joten se ei ehtinyt tähän palautukseen. Tästä syystä jotain piirtoartefakteja saattaa vielä esiintyä. Aika- ja tilavaativuuteen tämä ei kuitenkaan vaikuta.

Lisäksi BSP-puuta voi nopeuttaa monella eri tavoilla. Tilan voi esimerkiksi jakaa pienempiin osiin, jolloin niitä osia, jotka eivät näy kuvassa ollenkaan ei tarvitse käydä läpi lainkaan. Myös juurisolmun polygonin valinta vaikuttaa paljon BSP-puun muotoon, jonka tulisi olla mahdollisimman tasapainoinen. Tähänkin on olemassa algoritmeja, jotka selvittävät sopivia ehdokkaita ensimmäiseksi polygoniksi.