3D

Kolmemõõtmeliste lahenduste loomiseks on mitmeid raamistikke loodud, kus kasutajal võimalus kujundid ja kaamera sobivasse kohta sättida ning sealtkaudu meeldivat ruumilist pilti vaadata. Veebigraafika üks abiline näiteks three.js nime all. Samas lihtsamat ruumilisuse efekti kannatab täiesti ka harilike kahemõõtmeliste joonistusvahendite abil tekitada

Kaugemal väiksemaks

Arvutijooniste puhul tuleb ikka eristada ekraani- ja maailmakoordinaate. Päris lihtsate jooniste korral võib ette kujutada, et üks piksel ekraanil on üks meeter looduses. Või siis üks piksel vastab inimese pikkuse ühele sentimeetrile. Kõiksugu muude suurenduste puhul aga peab väärtused sobiva koefitsiendiga läbi korrutama. Samuti ei pruugi me tahta arve lugeda joonistusala vasakust ülanurgast, vaid mugavam võib olla arvutada mõnest rohkem joonise keskel olevast punktist - selle väärtuse saab arvutuste juures juurde liita. Matemaatikaõpikus oleme harjunud, et y-telg suundub üles, arvutiekraanil suundub see esmalt alla. Ka selle saab märgimuutusega paika sättida. Kolmanda mõõtme juures tuleb maailmakoordinaatidest ekraanikoordinaatideks arvutades lihtsalt üks tehe juurde - x ja y tuleb kaugusega (z) läbi jagada. Ehk siis sama suur asi kaks korda kaugemal näeb kaks korda väiksem välja. Sobivaks nägemiseks tulevad arvud suurenduskordajaga läbi korrutada. Siin puhul selleks valitud kümme. Muutujad keskx ja kesky tähistavad koordinaatide alguspunkti joonisel. Funktsioone ex ja ey kasutatakse ekraani x-i ja y-i arvutamiseks vastavalt maailmakoordinaatides punkti kolme mõõtme väärtustele.

```
var keskx=200;
var kesky=150;
var suurendus=10;
function ex(px, py, pz){ //x ekraanil
   return keskx+suurendus*px/pz;
}

function ey(px, py, pz){
   return kesky-suurendus*py/pz;
}
```

Joonistamise puhul tuleb siis iga kord sobivas kohas lihtsalt vastavad funktsioonid välja kutsuda

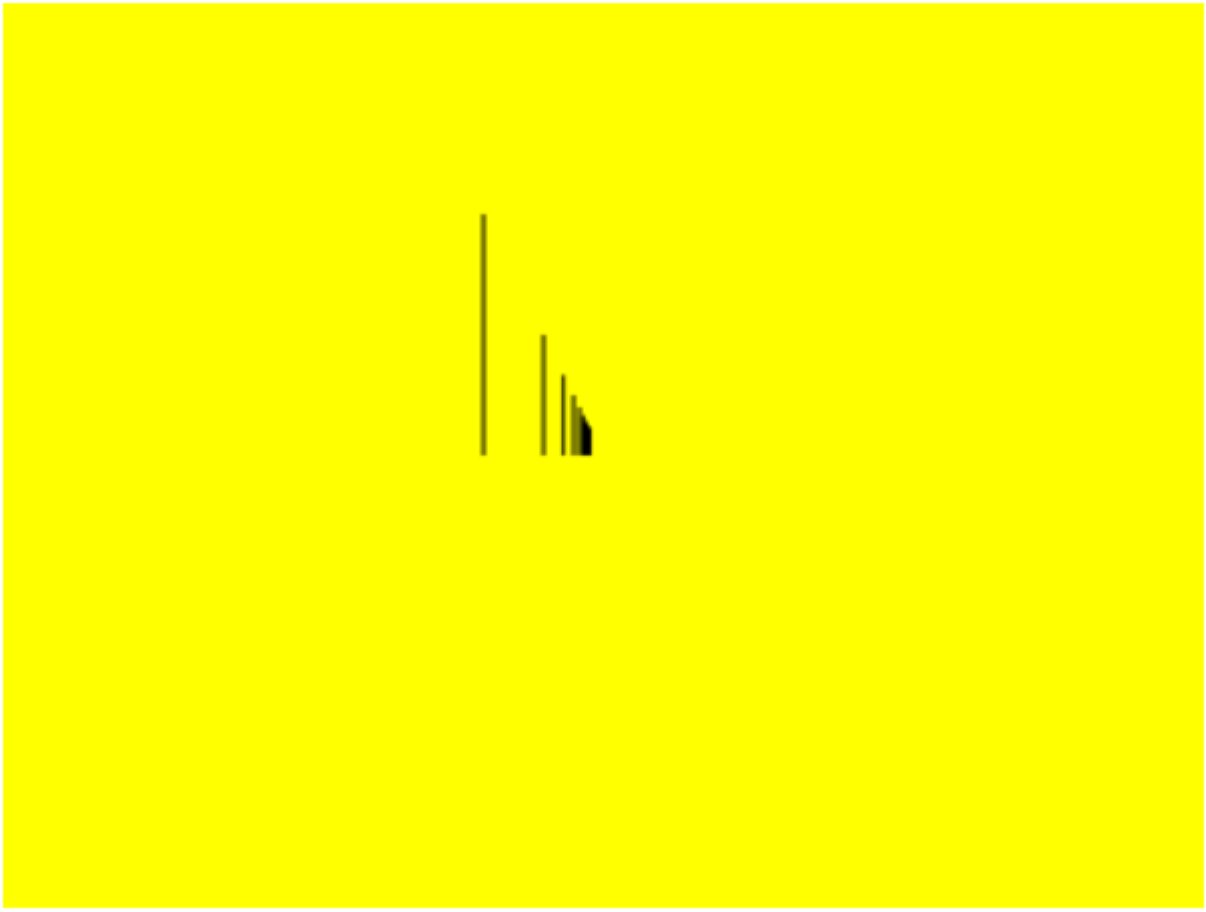
```
g.lineTo(ex(x, y, z), ey(x, y, z));
```

Ehk siis leitakse kolmemõõtmelistele algandmetele vastavad kahemõõtmelised punktid ekraanil. Joonisele tehakse tsükli abil postirida. Posti alumise otsa y on 0, ülemise oma 40 ühikut. Kaugused viiest viiekümne ühikuni.

```
<!doctype html>
<html>
  <head>
    <title>3D</title>
    <script>
       var keskx=200;
       var kesky=150;
       var suurendus=10;
       function ex(px, py, pz){ //x ekraanil
          return keskx+suurendus*px/pz;
       }

       function ey(px, py, pz){
          return kesky-suurendus*py/pz;
```

```
      }

      function joonista(){
        var g=document.getElementById("tahvel1").getContext("2d");
        g.beginPath();
        var x=-20;
        var y=40;
        for(var z=5; z<50; z+=5){
           g.moveTo(ex(x, 0, z), ey(x, 0, z));
           g.lineTo(ex(x, y, z), ey(x, y, z));
        }
        g.stroke();
      }
    </script>
  </head>
  <body onload="joonista();">
     <canvas id="tahvel1" width="400" height="300"
      style="background-color: yellow"/></canvas>
  </body>
</html>
```

Väikese ettekujutuse tulemusena võib lehel täiesti viisakat postirida näha.



Ülesandeid

- Pane näide tööle
- Katseta postide mitmesuguste kõrguste ja vahedega
- Loo sarnane postirida ka vaatajast paremale
- Paiguta postide otsa väikesed ringid nagu tänavalaternad

Kasutaja asukoha muutmine

Kolmemõõtmelises graafikas võivad liikuda kujundid, võib liikuda ka aga kaamera ehk vaataja ise. Kui vaatesuund jääb otse ette, siis polegi arvutamisel muud muret, kui tuleb ainult kasutaja asukoht kujundite asukohast maha lahutada. Ekraani x-koordinaadi arvutamine siis:

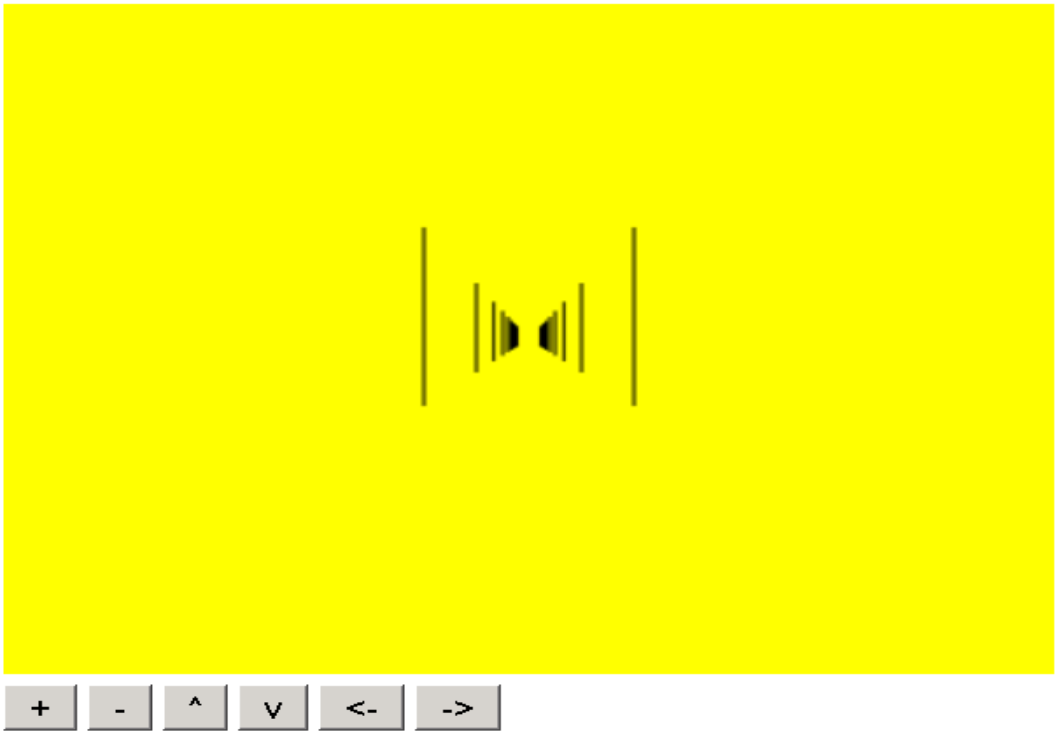
```
function ex(px, py, pz){ //x ekraanil
   return keskx+suurendus*(px-vaatajax)/(pz-vaatajaz);
}
```

Ehk siis nii x- kui z- koordinaadi puhul on vastav tehe tehtud. Mida ettepoole ise liikuda, seda lähemale tulevad kujundid. Mida ülespoole minna, seda enam paistavad kujundid allpool. Vaataja sammu muutuja näitab, kui pikkade hüpetega ta liigub. Juurde nupuriba vaataja andmete muutmiseks ning tekibki juba julgem ruumilise vaatamise tunne.

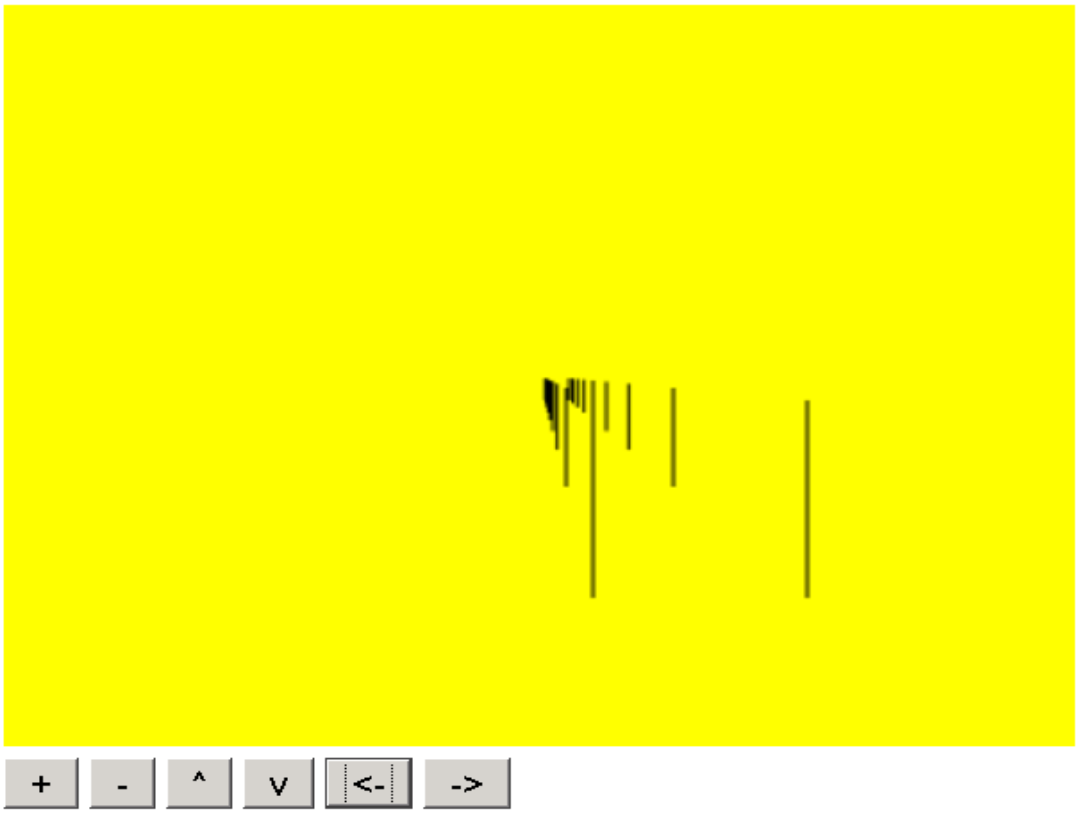
```
<!doctype html>
<html>
  <head>
    <title>3D</title>
    <script>
       var keskx=200;
       var kesky=150;
       var suurendus=10;
       var vaatajax=0;
       var vaatajay=0;
       var vaatajaz=-30;
       var vaatajasamm=5;
       function ex(px, py, pz){ //x ekraanil
          return keskx+suurendus*(px-vaatajax)/(pz-vaatajaz);
       }

       function ey(px, py, pz){
          return kesky-suurendus*(py-vaatajay)/(pz-vaatajaz);
       }

       function joonista(){
         var g=document.getElementById("tahvel1").getContext("2d");
         g.clearRect(0, 0, 400, 300);
         g.beginPath();
         var x=-20;
         var y=40;
         for(var z=5; z<50; z+=5){
            g.moveTo(ex(x, 0, z), ey(x, 0, z));
            g.lineTo(ex(x, y, z), ey(x, y, z));
         }
         //Parema poole postid
         x=20;
         for(var z=5; z<50; z+=5){
            g.moveTo(ex(x, 0, z), ey(x, 0, z));
            g.lineTo(ex(x, y, z), ey(x, y, z));
         }
         g.stroke();
       }

       function vasakule(){vaatajax-=vaatajasamm; joonista();}
       function paremale(){vaatajax+=vaatajasamm; joonista();}
       function yles(){vaatajay+=vaatajasamm;     joonista();}
       function alla(){vaatajay-=vaatajasamm;     joonista();}
```

```
      function edasi(){vaatajaz+=vaatajasamm;    joonista();}
      function tagasi(){vaatajaz-=vaatajasamm;   joonista();}
    </script>
  </head>
  <body onload="joonista();">
     <canvas id="tahvel1" width="400" height="300"
      style="background-color: yellow"/></canvas><br />
     <input type="button" onclick="edasi()" value="+"/>
     <input type="button" onclick="tagasi()" value="-"/>
     <input type="button" onclick="yles()" value="^"/>
     <input type="button" onclick="alla()" value="v"/>
     <input type="button" onclick="vasakule()" value="&lt;-"/>
     <input type="button" onclick="paremale()" value="-&gt;"/>

  </body>
</html>
```

Vaade eest



ning vaade pärast mõningast üles- ja vasakule poole liikumist.



Ülesandeid

- Pane näide tööle
- Koosta kujundiks nelinurkse varjualuse sõrestik - postid ja katusepüramiid. Vaata seda mitmest asukohast
- Koosta kujundiks kuuenurkse varjualuse sõrestik, vaata
- Koosta kujundiks kasutaja määratud nurkade arvuga varjualuse sõrestik

Kolmemõõtmeliste andmete objekt

Üksikuid punkte ja jooni arvutada ja kuvada saab täiesti tavaliste muutujatega. Kui aga vaja koos hakata liigutama hulgast punktidest koosnevaid kujundeid, siis tekib üksikuid punkte tülikalt suurel hulgal. Objektitüüpide ja objektide abil aga on võimalik neid mugavamalt hallata, neile ühekorraga käsklusi jagada. Samuti saab niimoodi mitmemõõtmelised arvutustehted nõnda koodi sisse ära peita, et hilisema töö juures ei pea nii palju arvutusvigade pärast muretsema, kui kood algul korralikult üle kontrollitud sai.

Vektor

Punkt ja vektor mõnevõrra sarnased nähtused. Esimesega küll oleme ehk enam harjunud tähistama asukohta ning teisega liikumist. Aga iseenesest mõlemal on vajalik arv mõõtmeid ning saab külge panna arvutusteks vajalikud käsud, nii et otsest põhjust eraldi tüüpide loomiseks ei ole. Mõistet asukohavektor ka täiesti kasutatakse, nii et las siis siingi piirdutakse ühe tüübiga kolme koordinaadiga määratud suuruse tähistamiseks. Juurde levinud funktsioonid liitmise ja lahutamise kohta. Samuti sisu tekstina väljastamiseks. Kusjuures tähele tasub panna, et praeguse kahe vektori liitmisel tekib uus, kolmas vektor, esialsete vektorite koordinaadid ei muutu.

Objektidest andmete tekstina välja küsimiseks on Javaskriptil olemas mugav JSON (JavaScript Object Notation) - vorming. Andmed paigutatakse looksulgude vahele. Sinna sisse järgemööda tunnuse nimi(võti) ning kooloni taga järgnev väärtus. Nagu näiteks siin algandmed

```
{"x":2,"y":4,"z":6}
```

Käsklus JSON.stringify() teeb objektist sarnase andmetega stringi. Kui mõnikord on vaja andmeid tekstist tagasi terviklikuks objektiks muuta, siis selleks sobib käsklus JSON.parse().

Nüüd aga kolmemõõtmelise vektori kood

```
<!doctype html>
<html>
  <head>
    <title>3D</title>
    <script>
       function Vektor3D(x, y, z){
          this.x=x;
          this.y=y;
          this.z=z;
          this.liida=function(v2){
            return new Vektor3D(this.x+v2.x, this.y+v2.y, this.z+v2.z);
          }
          this.lahuta=function(v2){
            return new Vektor3D(this.x-v2.x, this.y-v2.y, this.z-v2.z);
          }
          this.tekstina=function(){
             return JSON.stringify(this);
          }
       }
       function alusta(){
          var v1=new Vektor3D(2, 4, 6);
          var v2=new Vektor3D(0, 1, 0);
          var v3=v1.liida(v2);
          document.getElementById("kiht1").innerHTML=v3.tekstina();
       }
    </script>
  </head>
  <body onload="alusta();">
    <div id="kiht1"></div>
  </body>
</html>
```

Töö tulemus kihil:

```
{"x":2,"y":5,"z":6}
```

Ülesandeid

- Pane näide tööle
- Lisa vektorile käsklus korruta(arv), mis väljastab uue vektori, kus kõik esialgse vektori koordinaadid on selle arvuga korrutatud. Veendu, et tulemus toimib.

Kahest punktist joon

Vektor hoiab ühe punkti kolm koordinaati ilusti koos. Kui tahta, et joone andmeid saaks ühe

tervikuna käsitleda, siis tuleb omakorda joone kaks otspunkti üheks tervikuks siduda. Siin seda ka tehakse. Joonele luuakse punktide jaoks massiiv, kus kohal 0 on üks otspunkt ning kohal 1 teine.

Arvestades varasemat näidet vaataja asukoha liigutamise kohta, lisatakse eraldi tüüp ja objekt ka vaataja asukoha meelespidamiseks ning vastavalt sellele Joone joonistuskäskluses ekraanikoordinaatide arvutamiseks. Valemid sarnased kui enne, lihtsalt nüüd on need funktsioonide sisse pandud ja funktsioonid omakorda objektiks kapseldatud, et juhtkoodis tekkida võivat segadust vähemaks saada. Kui tükid olemas, siis nende ühendamine võiks juba üsna lihtsalt minna. Praegu luuakse kõigepealt uus Joon, mille otspunktide koordinaatideks saavad kaks Vektor3D-d oma väärtustega. Muud toimetused pandi alusta-funktsiooni sisse, sest tahvlile pole võimalik lehe päises avanemise ajal ligi saada. Küll aga on tahvel kättesaadav pärast lehe laadimist body onload-sündmuse peale käivitatava funktsiooni alusta() sees. Vaatajale määratakse asukoht (hetkel nullpunkt), joonistamise graafiline kontekst ning koordinaatide nullkoha asukoht ekraanil (hetkel tahvli keskkoht). Pärast joonistuskäsklust võikski joon ekraanil näha olla.

```
var j1=new Joon(new Vektor3D(-20, 0, 5), new Vektor3D(-20, 40, 5));
var vaataja;
function alusta(){
   var tahvel=document.getElementById("tahvel1");
   vaataja=new Vaataja(new Vektor3D(0, 0, 0),
     tahvel.getContext("2d"), tahvel.width/2, tahvel.height/2);
   j1.joonista(vaataja);
}
```

Edasi töötav kood tervikuna.

```
<!doctype html>
<html>
  <head>
    <title>3D</title>
    <script>
       function Vektor3D(x, y, z){
          this.x=x;
          this.y=y;
          this.z=z;
          this.liida=function(v2){
            return new Vektor3D(this.x+v2.x, this.y+v2.y, this.z+v2.z);
          }
          this.lahuta=function(v2){
            return new Vektor3D(this.x-v2.x, this.y-v2.y, this.z-v2.z);
          }
          this.tekstina=function(){
             return JSON.stringify(this);
          }
       }

       function Joon(v1, v2){
          this.punktid=new Array();
          this.punktid[0]=v1;
          this.punktid[1]=v2;
          this.joonista=function(vaataja){
             vaataja.g.beginPath();
             vaataja.g.moveTo(
                  vaataja.ex(this.punktid[0]), vaataja.ey(this.punktid[0]));
             vaataja.g.lineTo(
                  vaataja.ex(this.punktid[1]), vaataja.ey(this.punktid[1]));
             vaataja.g.stroke();
          }
       }
```

```
      function Vaataja(asukoht, g, keskx, kesky){
         this.asukoht=asukoht;
         this.g=g;
         this.suurendus=10;
         this.keskx=keskx;
         this.kesky=kesky;
         this.ex=function(p){
            var kaugus=p.lahuta(this.asukoht);
            return keskx+this.suurendus*kaugus.x/kaugus.z;
         }
         this.ey=function(p){
            var kaugus=p.lahuta(this.asukoht);
            return kesky-this.suurendus*kaugus.y/kaugus.z;
         }
      }

      var j1=new Joon(new Vektor3D(-20, 0, 5), new Vektor3D(-20, 40, 5));
      var vaataja;
      function alusta(){
         var tahvel=document.getElementById("tahvel1");
         vaataja=new Vaataja(new Vektor3D(0, 0, 0),
           tahvel.getContext("2d"), tahvel.width/2, tahvel.height/2);
         j1.joonista(vaataja);
      }
    </script>
  </head>
  <body onload="alusta();">
     <canvas id="tahvel1" width="400" height="300"
      style="background-color: yellow"/></canvas>
   </body>
</html>
```

Ülesandeid

- Pane näide tööle
- Joonista nõnda ekraanile kaks Joont
- Lisa Joonele värvi omadus
- Joonista ekraanile kaks eri värvi joont
- Muuda Vaataja asukohta

Joonte kogum

Üksikut joont on tõenäoliselt lihtsam ilma igasuguste objektideta joonistada. Kui mitu joont aga moodustavad uue terviku, siis on see mõistlik uude kesta panna - nii on võimalik seda hiljem korraga joonistada, liigutada või värvida. Kogumi juurde tuleb massiiv millesse andmeid lisada. Kogumile antud joonistuskäsklus käib läbi kõik kogumis olevad jooned ning kutsub välja neist igaühe joonistuskäskluse. Praegu siin juures ka käsklus vaataja tahvli puhastamiseks, aga selle saab mujale panna, kui tekib soov mitu kujundit järjestikku samale tahvlile joonistada.

```
function KujundiKogum(){
   this.kujundid=new Array();
   this.lisaKujund=function(k){this.kujundid.push(k);}
   this.joonista=function(vaataja){
      vaataja.g.clearRect(0, 0, 400, 300);
      for(var i=0; i<this.kujundid.length; i++){
         this.kujundid[i].joonista(vaataja);
      }
   }
}
```

Vaataja juurde sai liikumisoskus. Kõigepealt määrati vaatajale sammu pikkus, et kui palju ta iga korraga liigub.

```
this.vaatajasamm=5;
```

Liikumisfunktsiooni juures liidetakse vaataja asukohale sammuvektor.

```
this.liigu=function(samm){
   this.asukoht=this.asukoht.liida(samm);
}
```

Vastavalt liikumissuunale luuakse siis sammuvektor liikumiseks vaatajasammu jagu vastavas suunas.

```
this.paremale=function(){
        this.liigu(new Vektor3D(this.vaatajasamm, 0, 0));}
```

Lehele lisaks nupud vaataja liigutamiseks sobivas suunas. Kuna Vaataja-tüübist tehti vaataja-nimeline globaalmuutuja, siis saab sellele käsklusi andes lehel olevat vaadet muuta.

```
<input type="button" onclick="vaataja.edasi(); uuenda();" value="+"/>
```

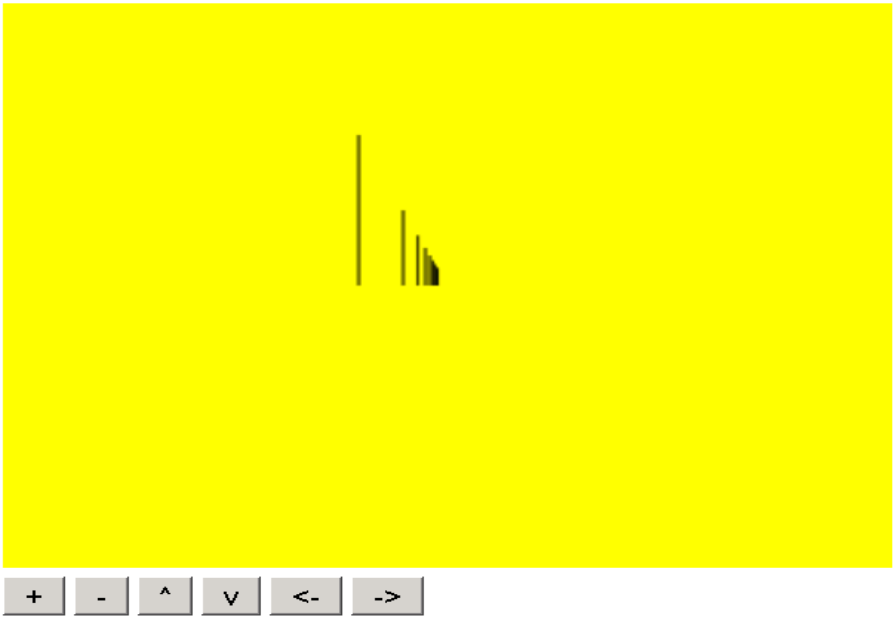
Edasi juba kood tervikuna.

```
<!doctype html>
<html>
  <head>
    <title>3D</title>
    <script>
```

```
function Vektor3D(x, y, z){
   this.x=x;
   this.y=y;
   this.z=z;
   this.liida=function(v2){
     return new Vektor3D(this.x+v2.x, this.y+v2.y, this.z+v2.z);
   }
   this.lahuta=function(v2){
     return new Vektor3D(this.x-v2.x, this.y-v2.y, this.z-v2.z);
   }
   this.tekstina=function(){
      return JSON.stringify(this);
   }
}

function Joon(v1, v2){
   this.punktid=new Array();
   this.punktid[0]=v1;
   this.punktid[1]=v2;
   this.joonista=function(vaataja){
      vaataja.g.beginPath();
      vaataja.g.moveTo(vaataja.ex(this.punktid[0]),
          vaataja.ey(this.punktid[0]));
      vaataja.g.lineTo(vaataja.ex(this.punktid[1]),
          vaataja.ey(this.punktid[1]));
      vaataja.g.stroke();
   }
}

function KujundiKogum(){
   this.kujundid=new Array();
   this.lisaKujund=function(k){this.kujundid.push(k);}
   this.joonista=function(vaataja){
      vaataja.g.clearRect(0, 0, 400, 300);
      for(var i=0; i<this.kujundid.length; i++){
         this.kujundid[i].joonista(vaataja);
      }
   }
}

function Vaataja(asukoht, g, keskx, kesky){
   this.asukoht=asukoht;
   this.g=g;
   this.suurendus=10;
   this.keskx=keskx;
   this.kesky=kesky;
   this.vaatajasamm=5;
   this.ex=function(p){
      var kaugus=p.lahuta(this.asukoht);
      return keskx+this.suurendus*kaugus.x/kaugus.z;
   }
   this.ey=function(p){
      var kaugus=p.lahuta(this.asukoht);
      return kesky-this.suurendus*kaugus.y/kaugus.z;
   }
   this.liigu=function(samm){
      this.asukoht=this.asukoht.liida(samm);
   }
   this.paremale=function(){
         this.liigu(new Vektor3D(this.vaatajasamm, 0, 0));}
   this.vasakule=function(){
         this.liigu(new Vektor3D(-this.vaatajasamm, 0, 0));}
   this.yles=function(){
         this.liigu(new Vektor3D(0, this.vaatajasamm, 0));}
```

```
            this.alla=function(){
                  this.liigu(new Vektor3D(0, -this.vaatajasamm, 0));}
            this.edasi=function(){
                  this.liigu(new Vektor3D(0, 0, this.vaatajasamm));}
            this.tagasi=function(){
                  this.liigu(new Vektor3D(0, 0, -this.vaatajasamm));}
         }

         var vaataja;
         var kogum;
         function alusta(){
            var tahvel=document.getElementById("tahvel1");
            vaataja=new Vaataja(new Vektor3D(0, 0, 0),
              tahvel.getContext("2d"), tahvel.width/2, tahvel.height/2);
            kogum=new KujundiKogum(vaataja);
            for(var z=5; z<50; z+=5){
              kogum.lisaKujund(
                new Joon(new Vektor3D(-20, 0, z), new Vektor3D(-20, 40, z)));
            }
            kogum.joonista(vaataja);
         }

         function uuenda(){
            kogum.joonista(vaataja);
         }
      </script>
   </head>
   <body onload="alusta();">
      <canvas id="tahvel1" width="400" height="300"
       style="background-color: yellow"/></canvas><br />
          <input type="button" onclick="vaataja.edasi(); uuenda();" value="+"/>
      <input type="button" onclick="vaataja.tagasi(); uuenda();" value="-"/>
      <input type="button" onclick="vaataja.yles(); uuenda();" value="^"/>
      <input type="button" onclick="vaataja.alla(); uuenda();" value="v"/>
      <input type="button" onclick="vaataja.vasakule(); uuenda();"
                value="&lt;-"/>
      <input type="button" onclick="vaataja.paremale(); uuenda();"
                value="-&gt;"/>
    </body>
</html>
```

Lehel saab nõnda vaatajaga ringi liikuda.



Ülesandeid

- Pane näide tööle
- Koosta joontest kaks kogumit: kahest joonest T-täht ning kuuest joonest tetraeeder (kolmnurkne püstprisma). Kuva mõlemad ekraanile. Vaata neid kaamera liigutamise abil mitmelt poolt.
- Lisa kogumile omaduseks toon. Kuva kumbki kogum vastava värviga ekraanile.

- Lisa kogumile käsklus teeSamm, mis siis kogumis olevaid kujundeid etteantud vektori jagu edasi liigutab. Lisa lehele nupp ühe ekraanil oleva kogumi liigutamiseks.
- Lisa kogumile muutuja liikumissammu vektorina hoidmiseks. Pane kumbki kujund ühtlase kiirusega tema küljes olevas suunas liikuma.

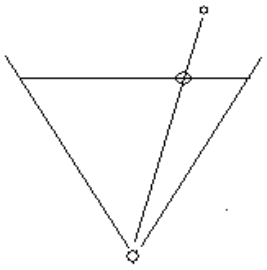
Reaalse mõõtkava arvestamine

Siiani tehtud näidetes jagati kaugustunnetuse saamiseks x ja y-koordinaat lihtsalt kaugusega läbi ning vajadusel korrutati suurenduskordajaga, et pilt enamvähem sobivas suuruses oleks. Tõsielulise olukorra järele tegemisel aga saab andmed ka täpsemalt näiteks meetrites välja arvestada ning siis vaateaval mõttelisel kaugusel olevale kilele projitseerida. Ning edasi otsustada, millise suurendusega projektsioonikile punktid arvutiekraani piksliteks ümber arvutada. Kolmnurga valemite järgi on vaadeldava punkti ühe (nt. x) mõõtme koordinaadi ja silmast kauguse suhe sama kui selle punktini viiva joone lõikekoha kilel ja kile kauguse suhe. Sellise arvutuse järgi õnnestub leida koht kilel

```
(px-vaatajax)*kilekaugus/(pz-vaatajaz)
```

Kuna vaataja ei pruugi olla koordinaatide nullpunktis ning tavajuhul on vaatekile mõõtmed pigem võrreldav ekraanipikslite arvuga sentimeetrites, siis saab kolmemõõtmelised koordinaadid kahemõõtmelisteks arvutada järgnevalt.

```
var suurendus=100; //sentimeetrit meetri kohta
function ex(px, py, pz){ //x ekraanil
   return keskx+suurendus*(px-vaatajax)*kilekaugus/(pz-vaatajaz);
}
```

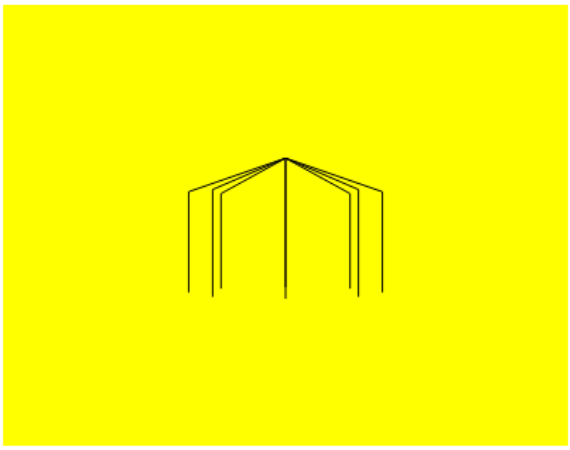


Edasi juba programmikood tervikuna

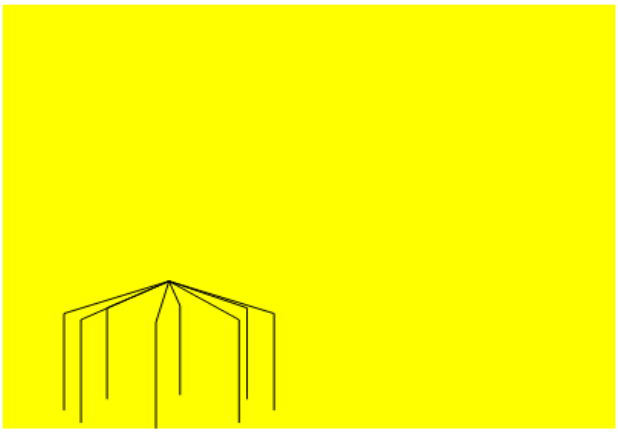
```
<!doctype html>
```

```
<html>
  <head>
    <title>3D</title>
    <script>
       var keskx=200;
       var kesky=150;
       var suurendus=100; //sentimeetrit meetri kohta
       var vaatajax=0;
       var vaatajay=2;
       var vaatajaz=-25;
       var kilekaugus=8; //meetrit
       var vaatajasamm=2;
       var telkx=0;
       var telkz=0;
       var telgiraadius=3; //meetrit
       var telgikorgus=3; //meetrit
       var tipukorgus=4; //meetrit
       var nurkadearv=8;
       function ex(px, py, pz){ //x ekraanil
          return keskx+suurendus*(px-vaatajax)*kilekaugus/(pz-vaatajaz);
       }

       function ey(px, py, pz){
          return kesky-suurendus*(py-vaatajay)*kilekaugus/(pz-vaatajaz);
       }

       function joonista(){
         var g=document.getElementById("tahvel1").getContext("2d");
         g.clearRect(0, 0, 400, 300);
         g.beginPath();
         var nurgavahe=2*Math.PI/nurkadearv;
         var nurk=0;
         for(var i=0; i<nurkadearv; i++, nurk+=nurgavahe){
            var x=telkx+telgiraadius*Math.cos(nurk);
            var z=telkz+telgiraadius*Math.sin(nurk);
            g.moveTo(ex(x, 0, z), ey(x, 0, z));
            g.lineTo(ex(x, telgikorgus, z), ey(x, telgikorgus, z));
            g.lineTo(ex(telkx, tipukorgus, telkz),
               ey(telkx, tipukorgus, telkz));
         }
         g.stroke();
       }

       function vasakule(){vaatajax-=vaatajasamm; joonista();}
       function paremale(){vaatajax+=vaatajasamm; joonista();}
       function yles(){vaatajay+=vaatajasamm;     joonista();}
       function alla(){vaatajay-=vaatajasamm;     joonista();}
       function edasi(){vaatajaz+=vaatajasamm;    joonista();}
       function tagasi(){vaatajaz-=vaatajasamm;   joonista();}
    </script>
  </head>
  <body onload="joonista();">
     <canvas id="tahvel1" width="400" height="300"
      style="background-color: yellow"/></canvas><br />
     <input type="button" onclick="edasi()" value="+"/>
     <input type="button" onclick="tagasi()" value="-"/>
     <input type="button" onclick="yles()" value="^"/>
     <input type="button" onclick="alla()" value="v"/>
     <input type="button" onclick="vasakule()" value="&lt;-"/>
     <input type="button" onclick="paremale()" value="-&gt;"/>
  </body>
</html>
```

Samuti kaheksanurkse telgi vaated mitmest asukohast.



+ - ^ v <- ->



+ - ^ v <- ->