Opdracht 2: WPF 3D Game

3D visualisatie in WPF en 2D fysische simulatie

Tim Vermeulen

10/10/2023

1 2D computer graphics
2 Basic game physics
3 Physics-based simulation
4 Inleiding 3D Graphics
5 3D in WPF
6 Opgave

1. 2D computer graphics

Basis begrippen

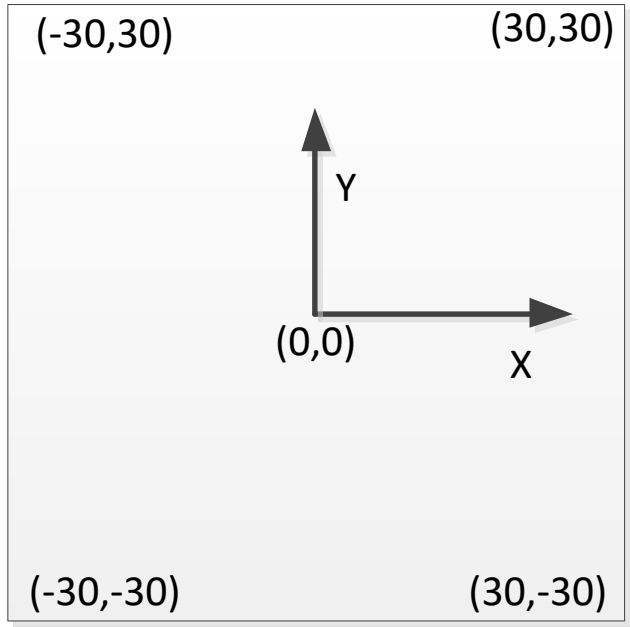
World Coördinate System (WCS)

Natuurlijk, logisch coördinatenstelsel voor de toepassing.

Voorbeeld: grafische weergave van de functie

$y(x)=x^2+x-6$ voor $x \in [-5, 5]$

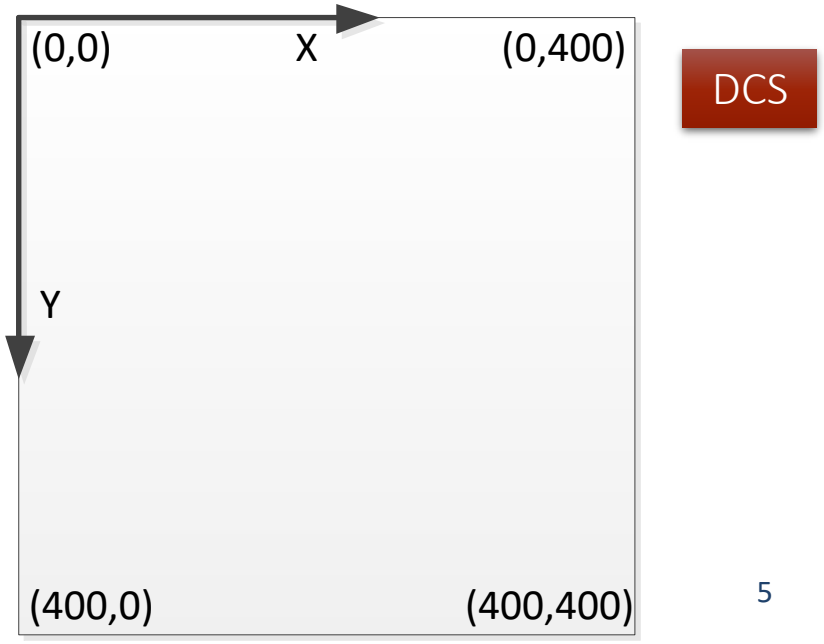
==> y is maximaal 24, mogelijke keuze:



WCS

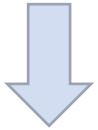
Device Coördinate System (DCS)

- Coördinatenstelsel zoals vastgelegd door het grafisch ‘device’:
- Voorbeeld: canvas van 400x400 pixels:



Coördinatentransformaties

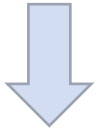
Onze code genereert logischerwijze coördinaten in WCS



Coördinaten in DCS (voor rendering).

Voorbeeld:

oorsprong (0,0) in WCS



omgerekend (200,200) in DCS

Omvorming moet gebeuren voor elke punt met coördinaten in WCS.

Coördinatentransformaties

Steeds minimaal twee delen:

- Schaling:

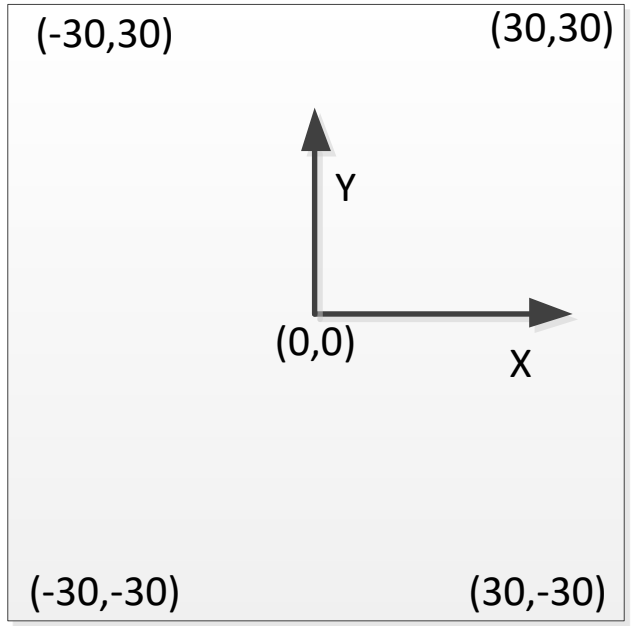
  X * 400 / 60 Y * 400/60 * (-1)

  (vanwege tegengestelde zin)

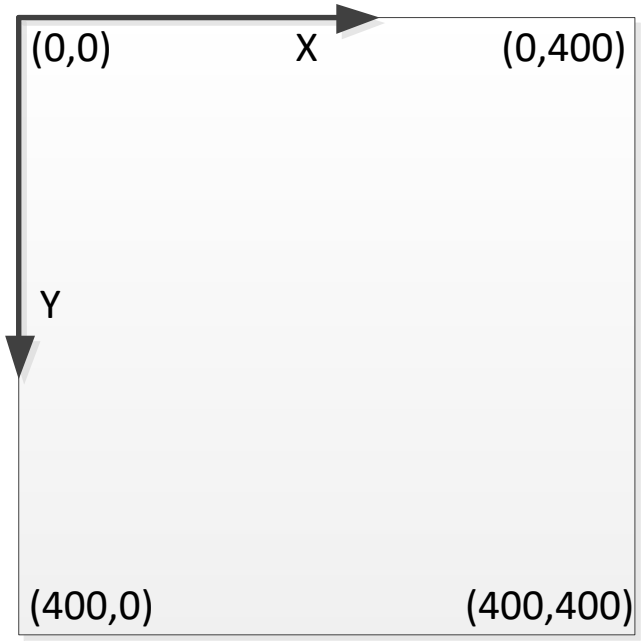
- Verplaatsing

  X + 30 Y - 30

- (soms rotatie)



WCS



DCS

Coördinatentransformaties

- Reeds voorzien in WPF:

```
var transform = new TransformGroup();
transform.Children.Add(new ScaleTransform(400/60, -400/60));
transform.Children.Add(new TranslateTransform(30, -30));
drawingCanvas.RenderTransform = transform;
```

2. Basic game Physics

- Basiswetten uit Fysica: beweging
- Werken in 2 dimensies, ontbinden van vectoren
- Krachten, impuls, wetten van Newton, gravitatie, arbeid/energie/vermogen, wrijving, botsingen

Kinematica

Rechtlijnige beweging

Positie in functie van snelheid:

$$v_{gem} = \frac{x_1 - x_0}{t_1 - t_0} = \frac{\Delta x}{\Delta t}$$

$$v(t) = \lim_{\Delta t \to 0} \frac{\Delta x}{\Delta t} = \frac{dx}{dt} \quad \text{of} \quad dx(t) = v(t).dt$$

$$\boldsymbol{x(t) = x_0 + \int_{t_0}^{t} v(t)dt}$$

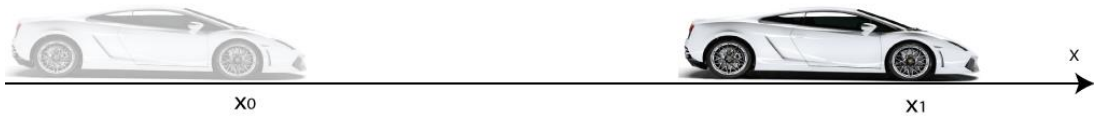
en bij constante snelheid en t_0=0 : $\boldsymbol{x(t) = x_0 + v.t}$

Differentievergelijking

Differentiaalvergelijking

Kinematica

Rechtlijnige beweging



versnelling:

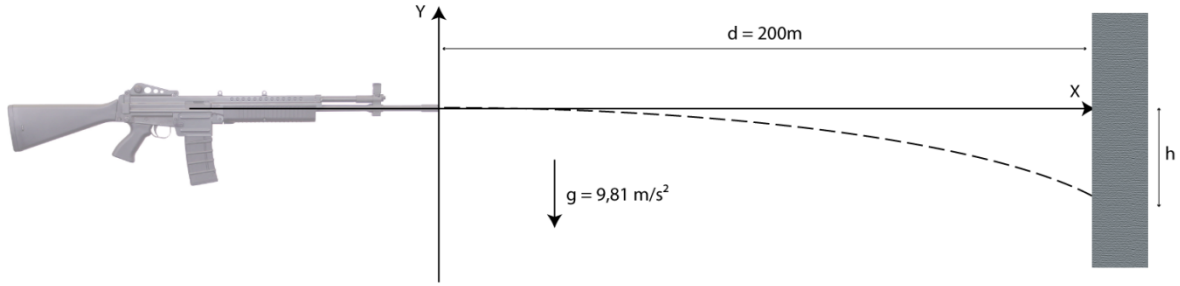
$$a_{gem} = \frac{v_1 - v_0}{t_1 - t_0} = \frac{\Delta v}{\Delta t} \qquad a(t) = \lim_{\Delta t \to 0} \frac{\Delta v}{\Delta t} = \frac{dv}{dt} = \frac{d^2x(t)}{dt^2}$$

$$v(t) = v_0 + \int_{t_0}^{t} a(t)dt$$ en bij constante versnelling en t_0=0: $v(t) = v_0 + a.t$

Combinatie: $$x(t) = x_0 + \int_{t_0}^{t} v(t)dt$$

$$\boldsymbol{x(t) = x_0 + v_0.t + \frac{a.t^2}{2}}$$

Lineaire beweging (2 dimensies)



Voorbeeld: een afgeschoten kogel

- *Ontbinden in 2 componenten (X, Y)*
- $a_x = 0, \qquad v_x = \frac{700m}{s}, \qquad t = \frac{200m}{v_x} = 0{,}286s$
- $a_y = -9{,}81m/s^2$ na *t:* $d_y = \frac{(a_y.t^2)}{2} = 0{,}4m$

De Wetten van Newton

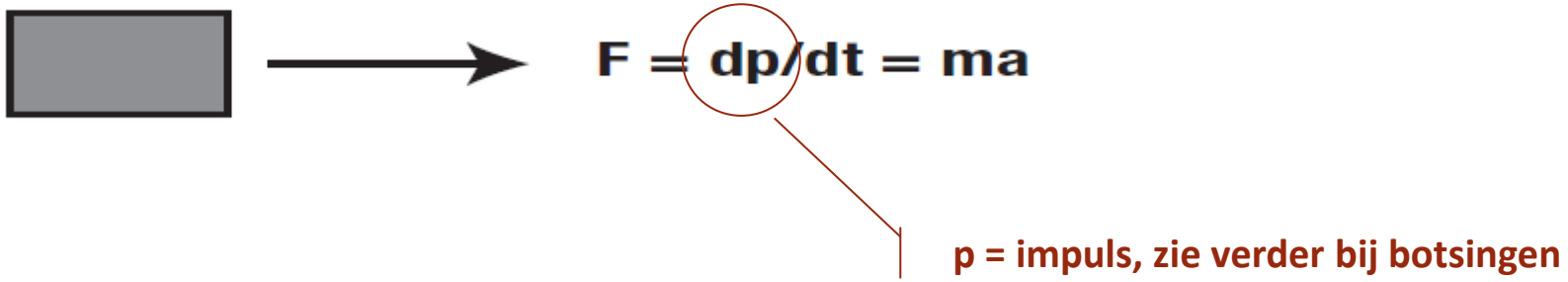
- **Wet 1** : Een vrij deeltje beweegt met constante snelheid zonder versnelling (traagheidswet, inertiewet)

$$\mathbf{F} = 0 \text{ implies } \mathbf{a} = 0$$

$$\mathbf{v} = \text{constant}$$

De Wetten van Newton

- **Wet 2** : De verandering van de beweging is evenredig met de kracht en volgt de rechte lijn waarin de kracht werkt

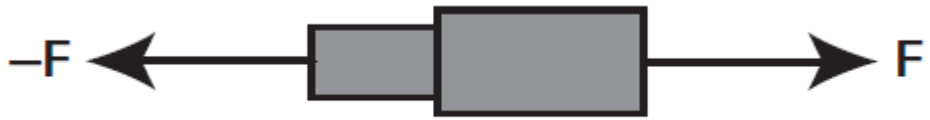


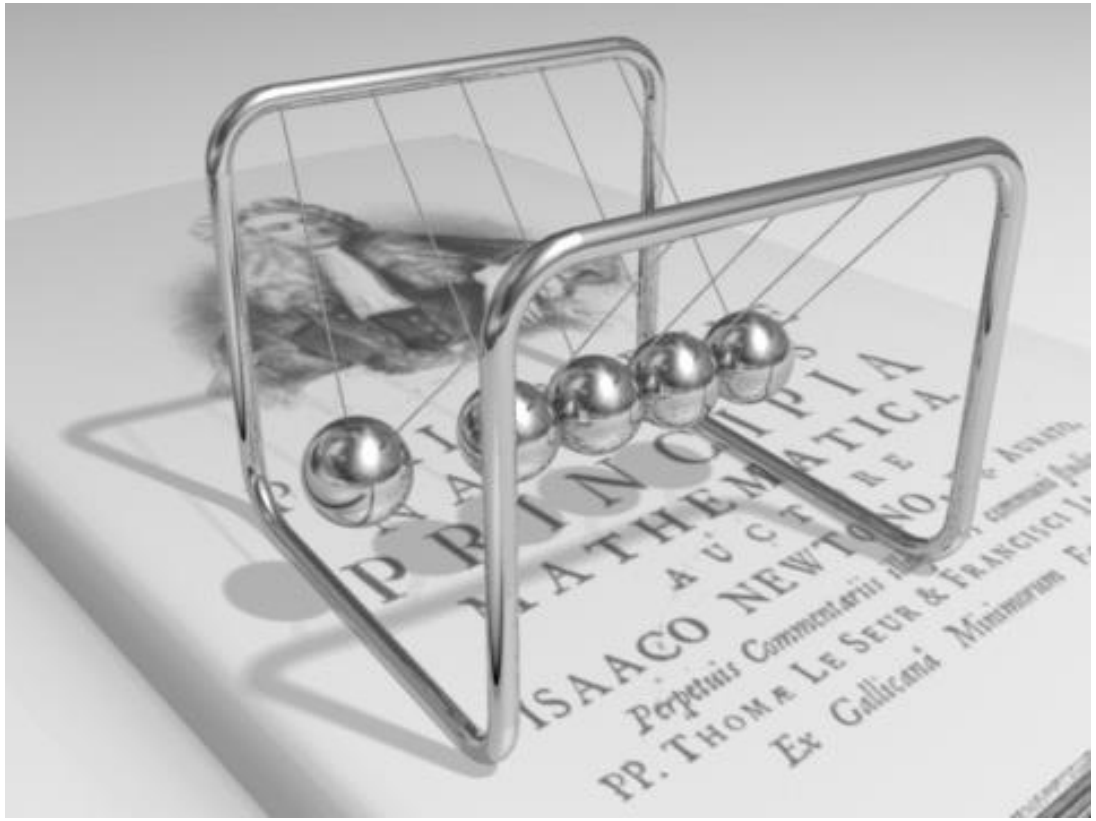
Massa, snelheid en kracht
De tweede wet van Newton

- Krachten zijn de oorzaak van beweging, vervorming, breking, …
- Eenheid van kracht : Newton : **1N=1kg m/s²**
- 1N geeft aan een massa van 1 kg een versnelling van 1m/s²
- Verschillende types: gravitatie, elektrische/magnetische aantrekking, wrijving, door druk die inwerkt op een lichaam (bv. in verbrandingsmotoren).
- massa van een voorwerp is een maat voor de kracht die nodig is om het voorwerp een bepaalde versnelling te geven:
- $$F = \frac{d}{dt}(m.v) \quad of \quad F = m\,.a$$ Als m onafhankelijk is van de tijd

De Wetten van Newton

- **WET 3** : Als een lichaam een kracht op een ander lichaam uitoefent, oefent dit laatste lichaam een even grote maar tegengestelde kracht uit op het eerste





Botsingen

Tussen botsende voorwerpen wordt impuls en energie uitgewisseld.

- Impuls is een alternatieve manier om de toestand van een object in beweging uit te drukken. De grootheid *m* . *v* wordt de impuls *p* (of hoeveelheid van beweging) genoemd $\vec{p} = m\,\vec{v}$
- En volgens 2de wet van Newton, bij cte massa $\vec{F} = \frac{d}{dt}(m.\vec{v}) = \frac{d}{dt}\vec{p}$
- **Wet van behoud van impuls**: in een geïsoleerd stelsel is de som van de impuls van alle deeltjes steeds constant

$$P = \sum_i p_i = p_1 + p_2 + p_3 + \cdots = Cte$$

Elastische Botsingen

- Bij elastische botsingen zijn zowel de totale impuls als de totale kinetische energie constanten van de beweging

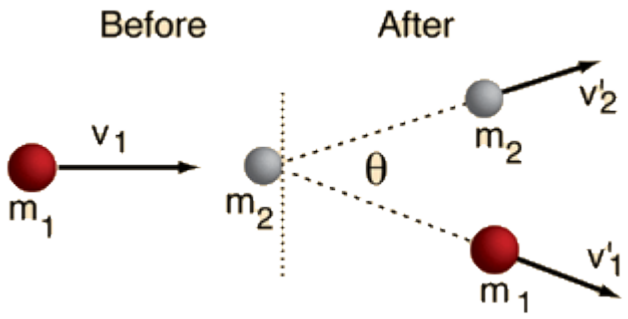
$$(m_1.\vec{v}_1 + m_2.\vec{v_2}) \; voor \; botsing = (m_1.\vec{v}_1 + m_2.\vec{v_2}) \; na \; botsing$$

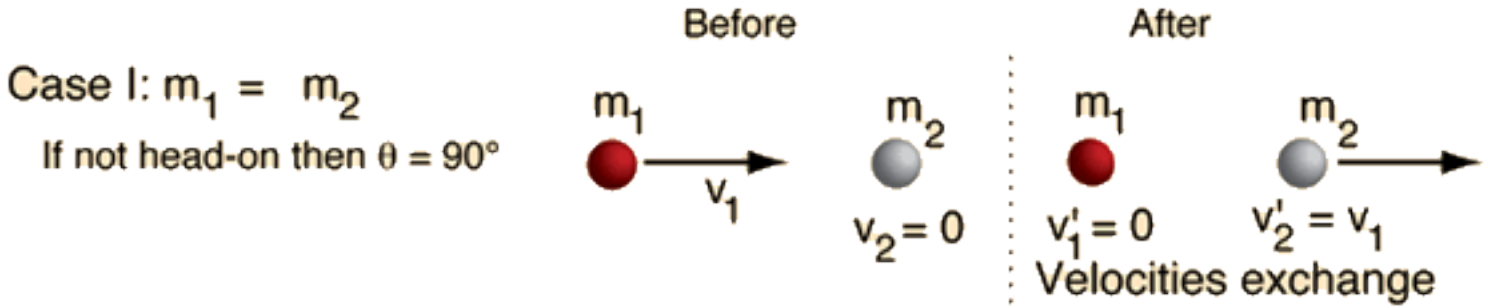
en

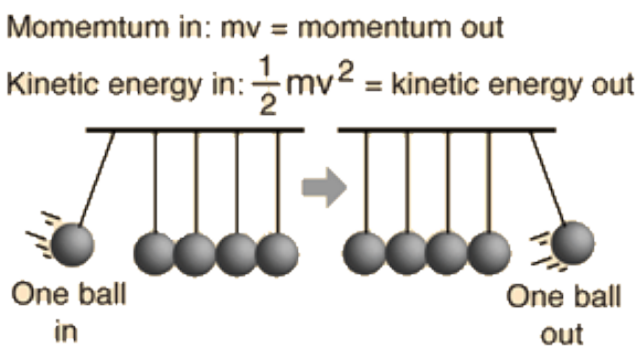
$$E_{kin} : \left(\frac{m_1.{v_1}^2}{2} + \frac{m_2.{v_2}^2}{2} \right) voor \; botsing = \left(\frac{m_1.{v_1}^2}{2} + \frac{m_2.{v_2}^2}{2} \right) na \; botsing$$

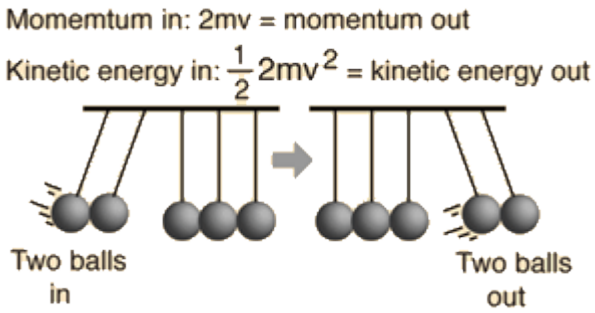
- Het verschil in snelheid voor en na de botsing is gelijk, maar de richting van de relatieve snelheid is omgekeerd

Elastische Botsingen









Niet perfect elastische botsingen

- De bal zal na elke botsing een lagere hoogte bereiken.
- Bij een niet elastische botsing zal de relatieve snelheid verminderen met factor e
- met voorwaarde: 0 < e < 1

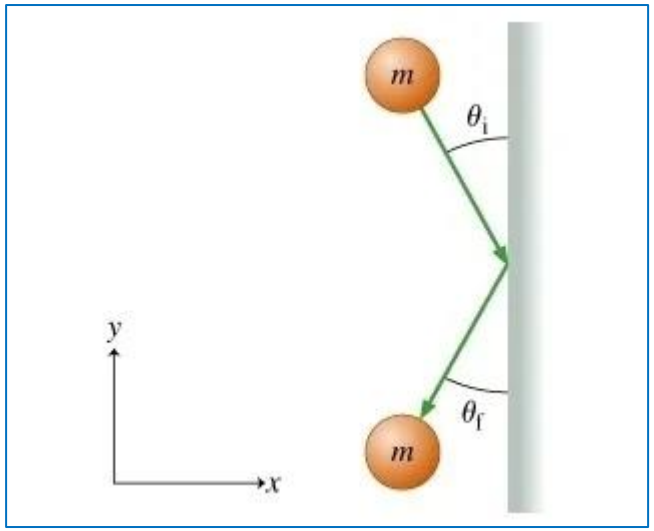
$$v_{1na} - v_{2na} = -e\left(v_{1voor} - v_{2voor}\right)$$

Botsingen – bal met wand

Snelheid kan ontbonden worden in een component evenwijdig met de muur en een component loodrecht op de muur.

- Evenwijdige component: botst niet ==> onveranderd
- Loodrechte component: keert om van richting

Wordt ook een '**reflectionvector**' genoemd



Massa vs Gewicht

- de gravitatiekracht: de aantrekkingskracht die voorkomt tussen alle massa's

$$F = G.\frac{m_1.m_2}{r^2}$$

met G = 6.67390e−11 N.m2/kg2.

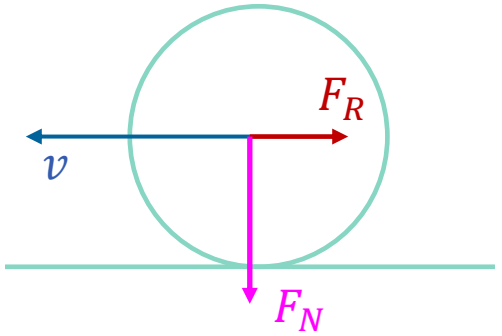
$$F = G.\frac{m_{aarde}}{r^2}.m = 6{,}67390.10^{-11}.\frac{5{,}9736.10^{24}}{(6{,}375.10^6)^2}.m = 9{,}81.m\ N$$

Valversnelling g=9,81 m/s²

- Massa = hoeveelheid materie
- Gewicht = de kracht die een lichaam uitoefent op een steunpunt (of ophanging) o.a. ten gevolge van de gravitatie
- Vb in de ruimte: Feather and hammer drop on the moon: http://www.youtube.com/watch?v=5C5_dOEyAfk

Kinetica - rolweerstand

Rolweerstand = kracht die beweging tegenwerkt als gevolg van vervorming, adhesie…



Formule: $F_R = C_{rr}.F_N$

met F_R = grootte van de rolweerstand,

C_{rr}= 'rolling resistance' coëfficient (afhankelijk van de materialen)

F_N = normaalkracht (gewicht) = massa *g met g = 9,81m/s²

Vertraging:

$$a = -1.\frac{F_R}{m} = -1.\frac{C_{rr}.m.g}{m} = -C_{rr}.g$$ (dus onafhankelijk van massa!)

'-' geeft aan dat 'versnelling' in tegengestelde richting van de snelheid werkt

Opgelet: enkel vertraging zolang er beweging is!

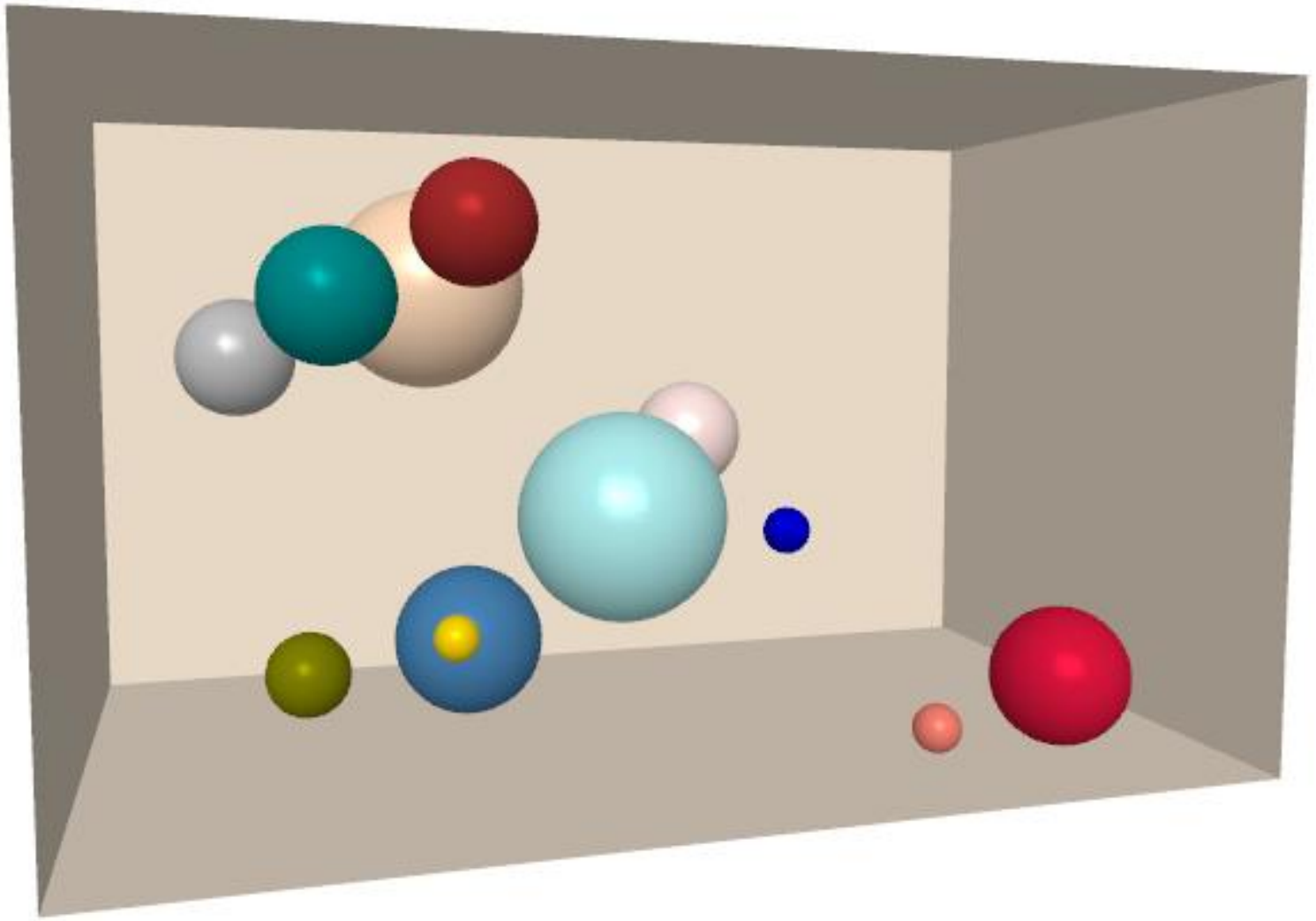
3. Physics-based simulatie

- Algemene structuur computergame/simulatie
- Case study: moving ball
- analytische methode vs. Euler

Algemene structuur computergame

- User interactie (sturen, acties, aanpassen parameters, manipulatie van objecten)
- Render-loop
  - Visualisatie van de actuele wereld
  - Genereert telkens één frame
  - Moet voldoende dikwijls uitgevoerd worden voor een aanvaardbare framerate (> 25Hz)
  - Bij gebruik van een framework zit deze meestal in het framework
- ‘Game Loop’
  - Collision detection: detecteren van interactie tussen objecten
  - Berekenen van de nieuwe toestand voor de objecten in de ‘wereld’ (op basis van bv. de wetten uit de fysica, AI)
- Timing: (voldoende nauwkeurig) timing-mechanisme is meestal vereist

Case Study: JoBotsingen.sln



analytische methode
Kinematica

Op voorhand alle (differentiaal) vergelijkingen oplossen en de positie van elk object vastleggen als functie van de tijd

- Bv.: eenparig versnelde lineaire beweging:

$$s(t) = s_0 + v_0.t + \frac{a.t^2}{2}$$

- In de game-loop op basis van de tijd de huidige positie berekenen voor elk object
- Als je interactie hebt van de speler is dit meestal niet mogelijk of zorgt dit voor een te grote vertraging

methode van EULER

De differentiaalvergelijkingen worden niet op voorhand opgelost, maar worden rechtstreeks gebruikt in de code om telkens de veranderingen in de versnelling, snelheid, positie enz… opnieuw te berekenen aan de hand van het tijdsinterval tussen twee berekeningen.

- Bvb Assumptie: tussen twee berekeningen is de snelheid constant

```
public void Integrate(particle p, float deltaT)
{
    p.positie += p.snelheid * deltaT;
    p.snelheid += p.versnelling * deltaT;
    p.versnelling = (1f / p.massa) * p.kracht;
}
```

methode van EULER

- veronderstelling kan te grote fout geven:
- veronderstel dat de versnelling a(t) tussen 2 frames constant blijft:

- In de ‘game-loop’ op basis van de tijd sinds vorig frame nieuwe positie, snelheid en eventueel versnelling berekenen.

```
public void Integrate(particle p, float deltaT)
{
    p.Positie += p.Snelheid * deltaT + (deltaT * deltaT / 2f) * p.Versnelling;
    p.Snelheid += p.Versnelling * deltaT;
    p.Versnelling = (1f / p.Massa) * p.Kracht;
}
```

methode van EULER

Probleem:
Er wordt nog steeds een fout gemaakt (vermits a(t) niet noodzakelijk constant is)
==> enkel geschikt voor kleine versnellingen/wijzigingen t.o.v Δt

- Voorbeeld: jet (v =1000Km/h)
  Bij een framerate van 100Hz is Δt=10ms en legt de jet tussen 2 frames 2,8m af, of na 1 sec 280m

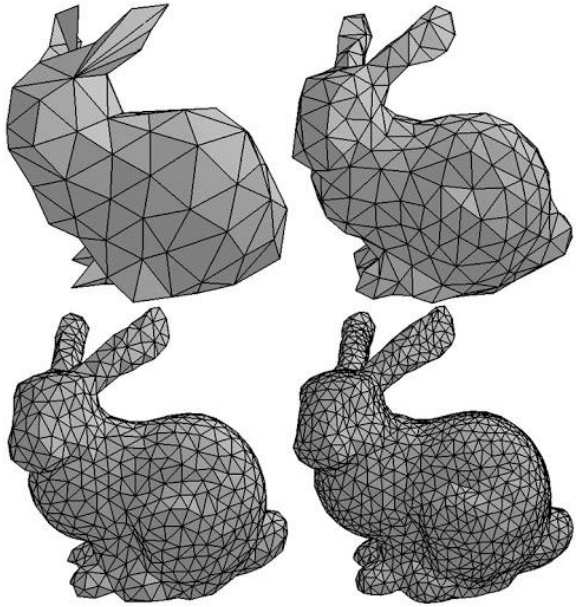
- Met een versnelling van 5m/s² wordt dit 285m. Als we de snelheid constant houden tussen 2 frames maken we dus een fout van 5m op 1 seconde!

Oplossing: 'methode van Runge Kutta' (stuk complexer)

4. Inleiding 3D Graphics

3D Graphics

- Objecten opgebouwd uit verzameling polygonen (meestal driehoeken).

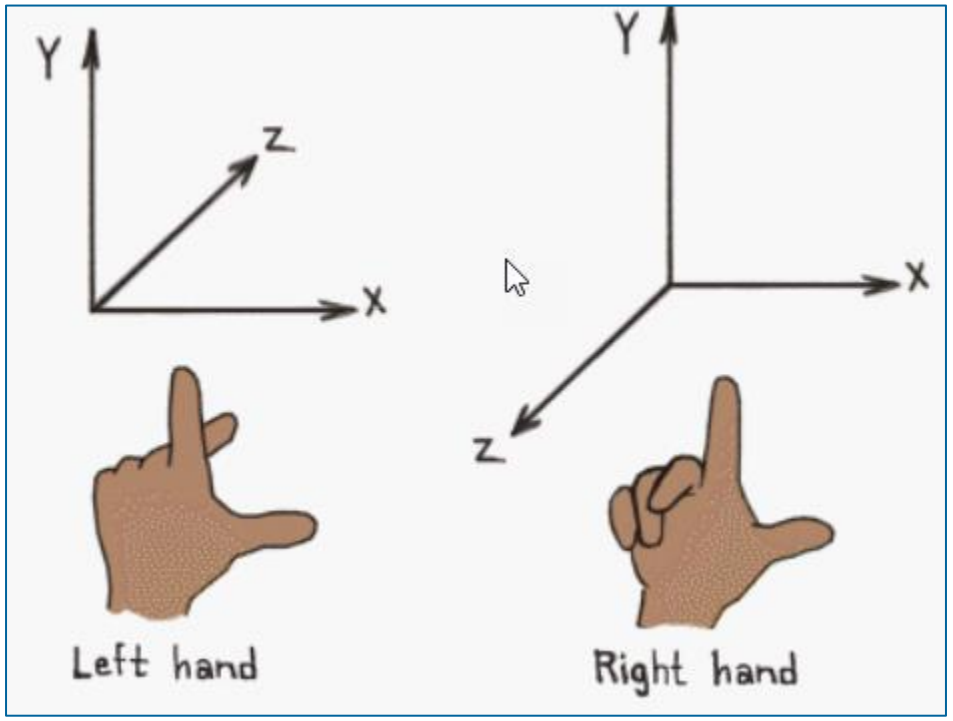


→ Lijst met punten (coördinaten)

→ Lijst met indexen van de hoekpunten (vertices)

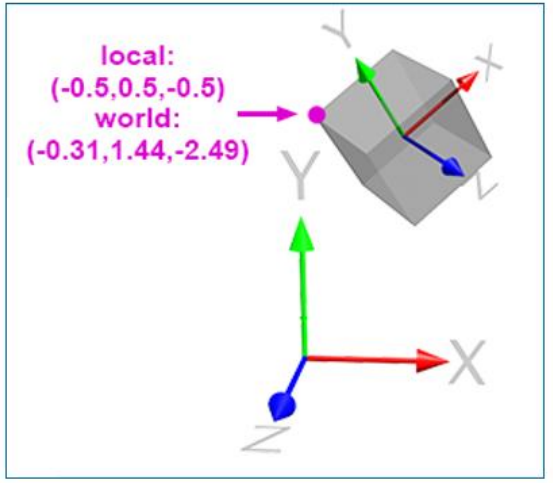
→ Afzonderlijke modellen: eigen lokaal coördinatenstelsel

World Coördinate System (WCS)



In WPF: rechtshandig WCS

Modellen: coördinaten LCS → WCS



Coördinatentransformaties

- Scaling, Rotation, Translation (volgorde is belangrijk!)
- Elke transformatie is voor te stellen via een transformatiematrix:

$$\begin{bmatrix} X_{new} \\ Y_{new} \\ Z_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \\ Z_{old} \\ 1 \end{bmatrix}$$

3D Scaling Matrix

$$\begin{bmatrix} X_{new} \\ Y_{new} \\ Z_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \\ Z_{old} \\ 1 \end{bmatrix}$$

3D Rotation Matrix

(For X-Axis Rotation)

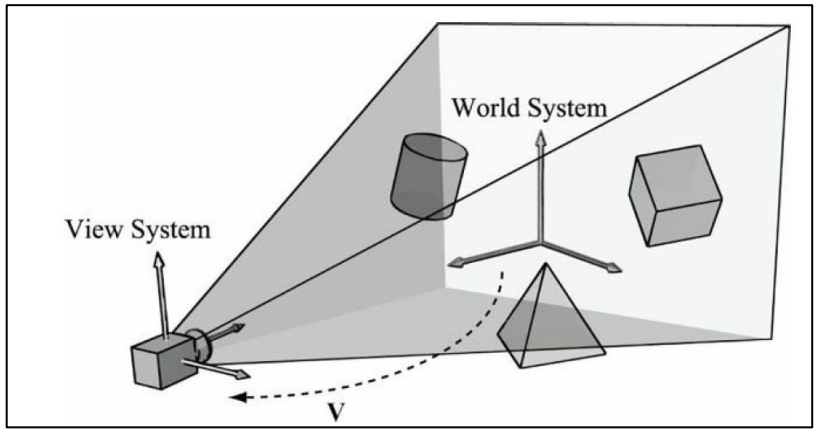
$$\begin{bmatrix} X_{new} \\ Y_{new} \\ Z_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \\ Z_{old} \\ 1 \end{bmatrix}$$

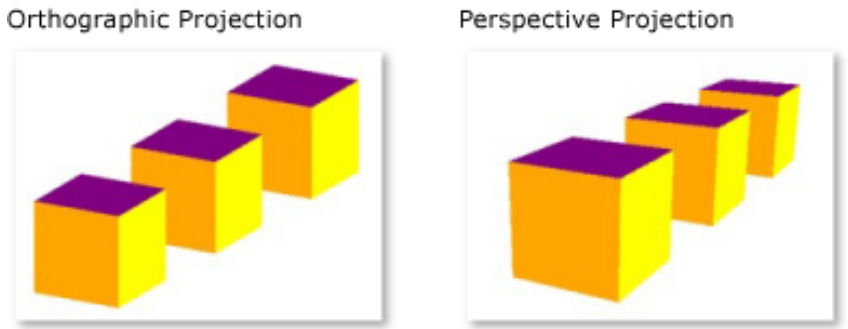
3D Translation Matrix

⇨ Gecombineerde transformatie: product van transformatiematrices

Camera

- Voor de projectie van de 3D-wereld op het 2D-scherm wordt het concept van een ‘camera’ gebruikt:





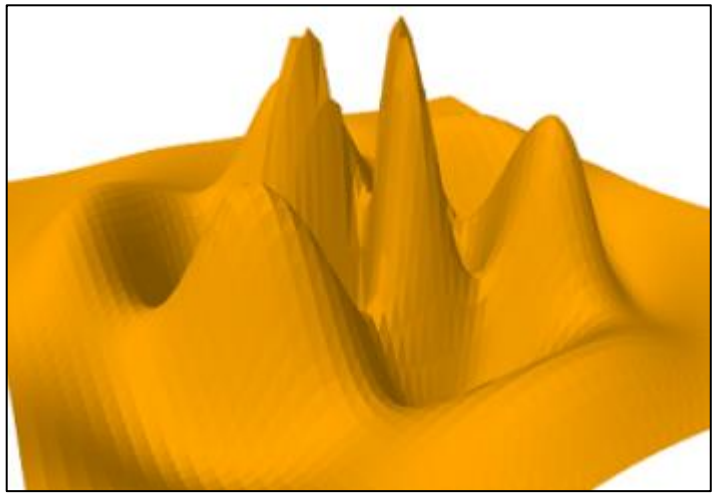
- Camera heeft een positie, kijkrichting en een ‘Up’-richting

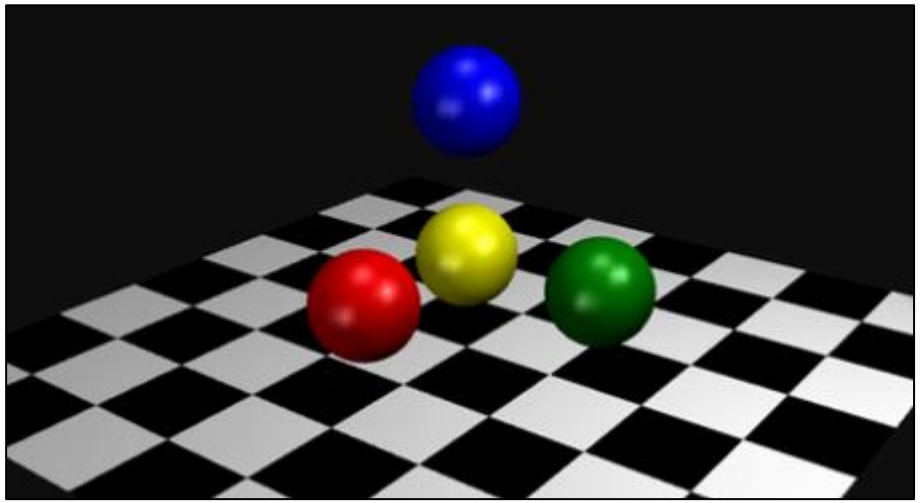
Belichting

- Licht geeft meer realistisch leven aan je 3D scene
- Licht betekent ook schaduw
- Verschillende lichtbronnen en parameters (lichtkleur, helderheid, schaduwmethoden, special effects, …) tunen

Belichting

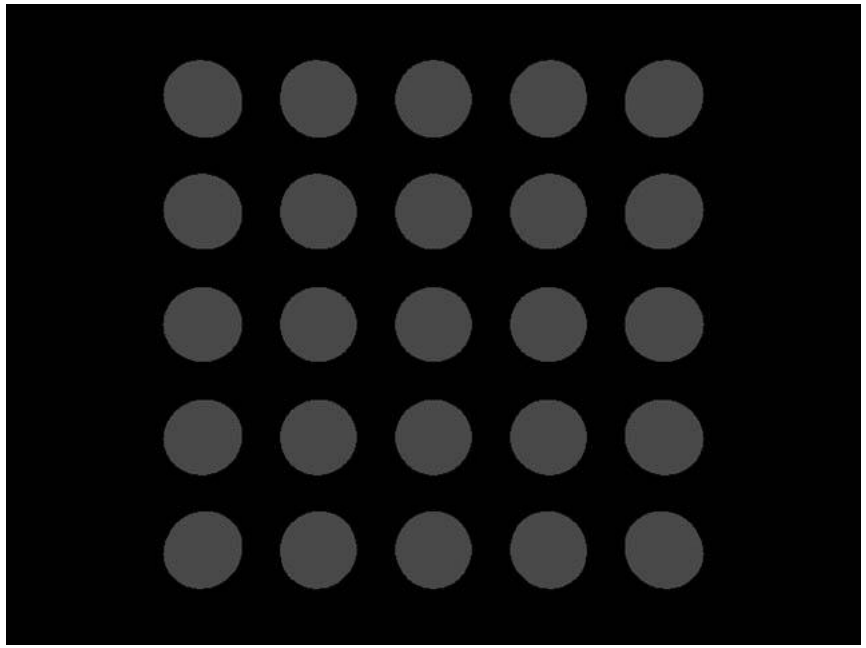
- Types lichtbronnen: Ambient, Directional, Point, Spot
  Meestal combinatie van ‘ambient’ met ander type.

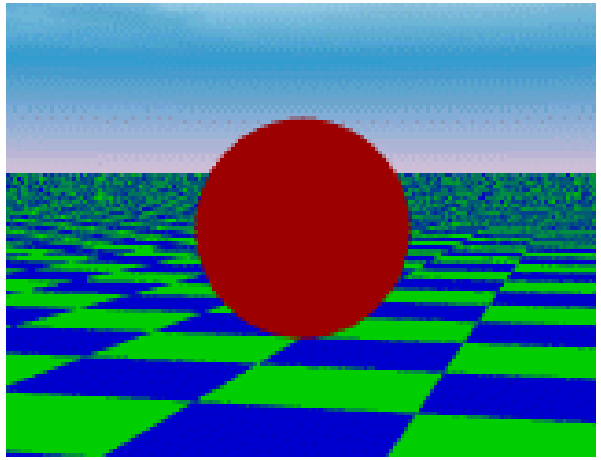




Ambient light

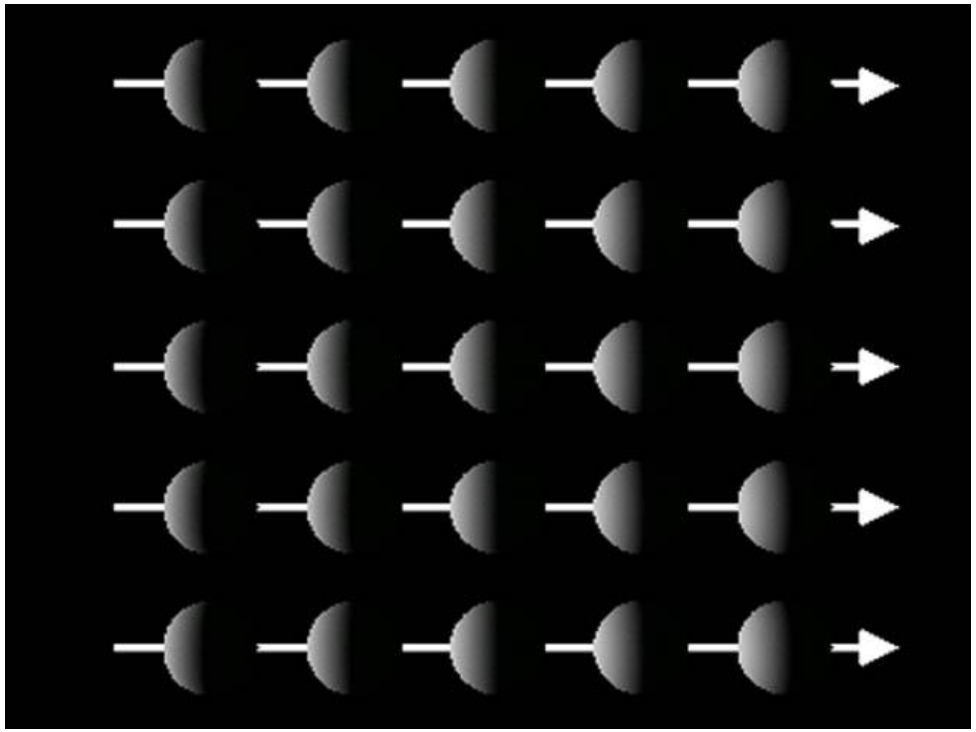
- Lichtbron zonder oorsprong of richting
- Het voordeel van een ambient light is dat je hele scène meteen verlicht is, het nadeel is echter dat er geen schaduwwerking optreedt waardoor alles een vrij steriele en “platte” indruk geeft.

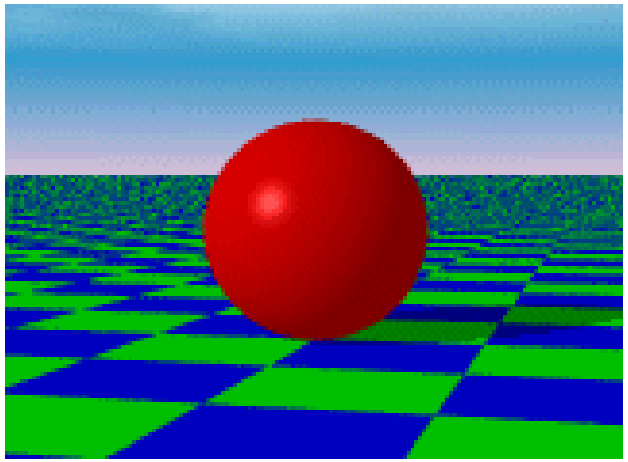




Directional light

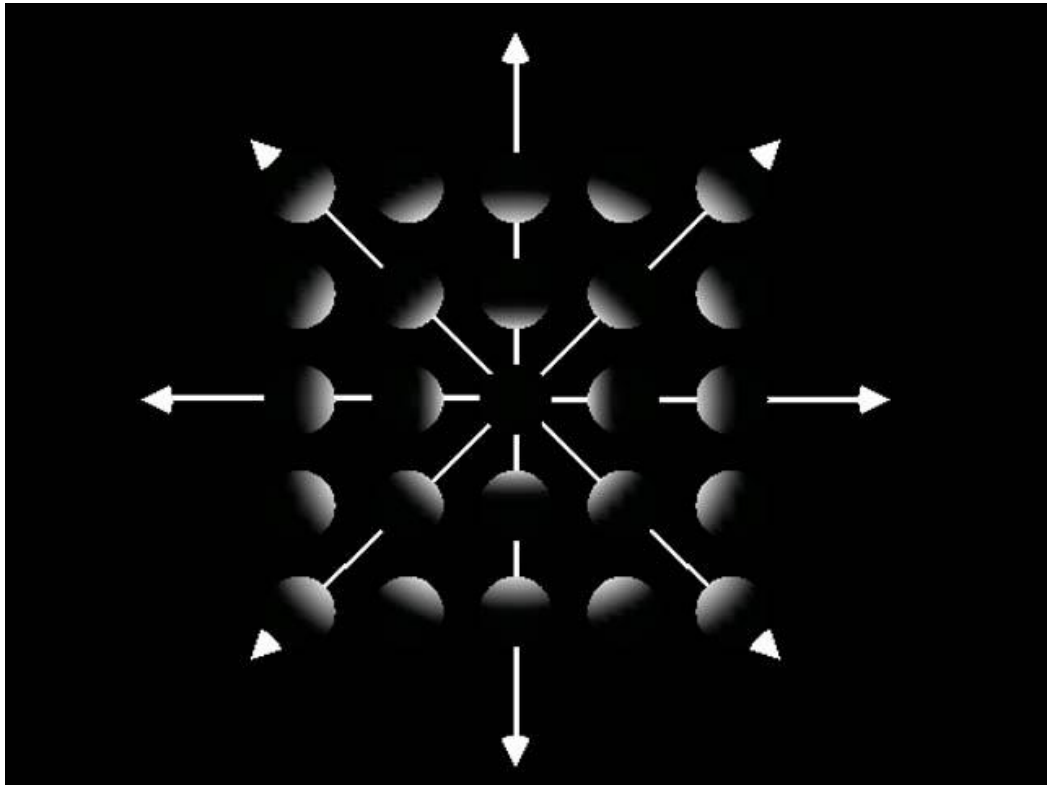
- Lichtbron op oneindig bvb de zon, de lichtstralen lopen evenwijdig

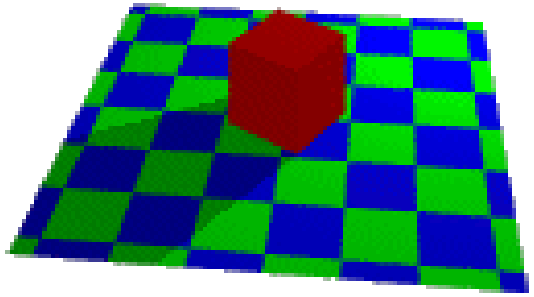




Point light

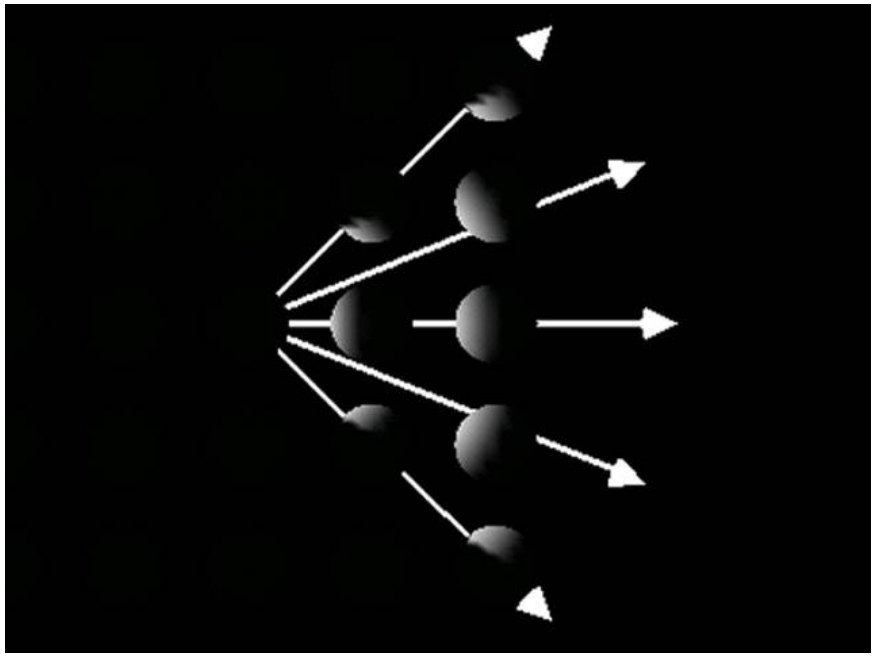
- Een puntlichtbron stuurt het licht in alle richtingen, vb een gloeilamp

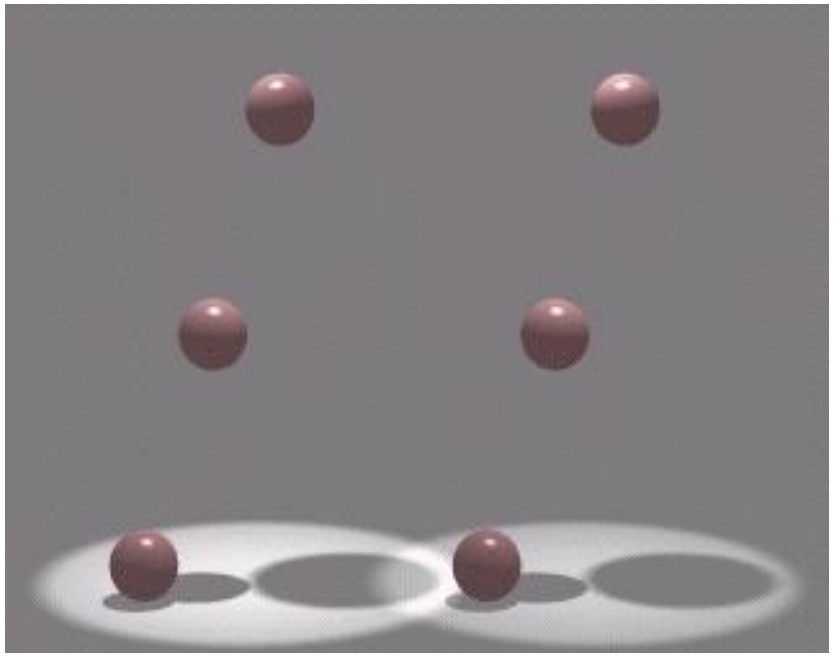




Spot light

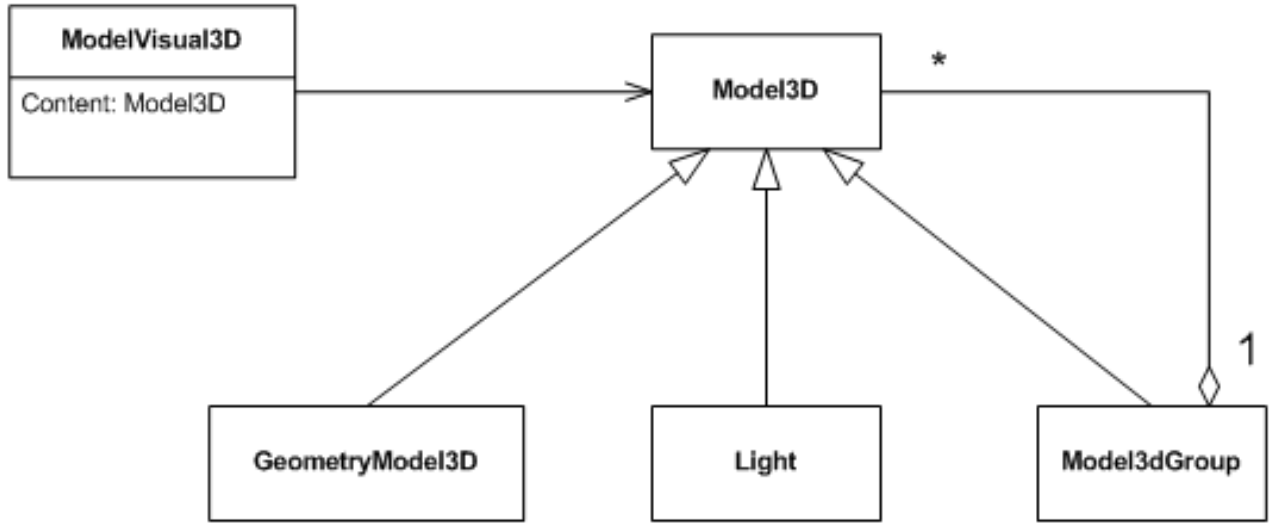
- De lichtintensiteit van de centrale kegelas neemt af





5. 3D Graphics in WPF

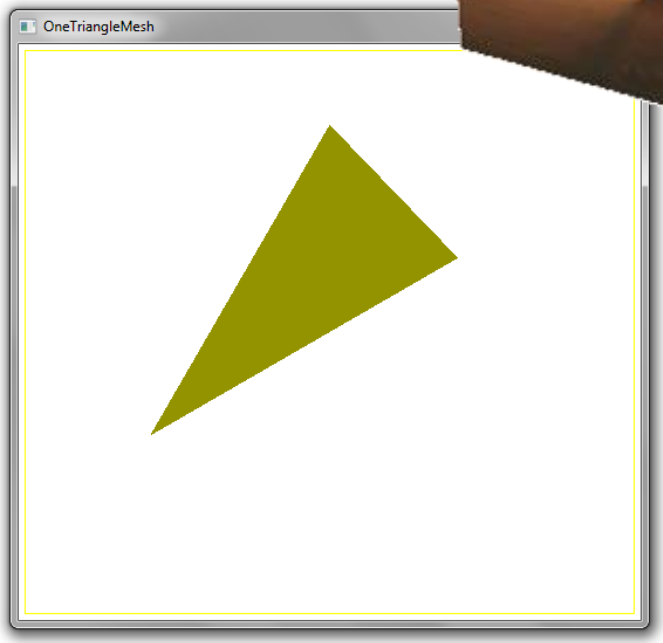
3D Graphics in WPF

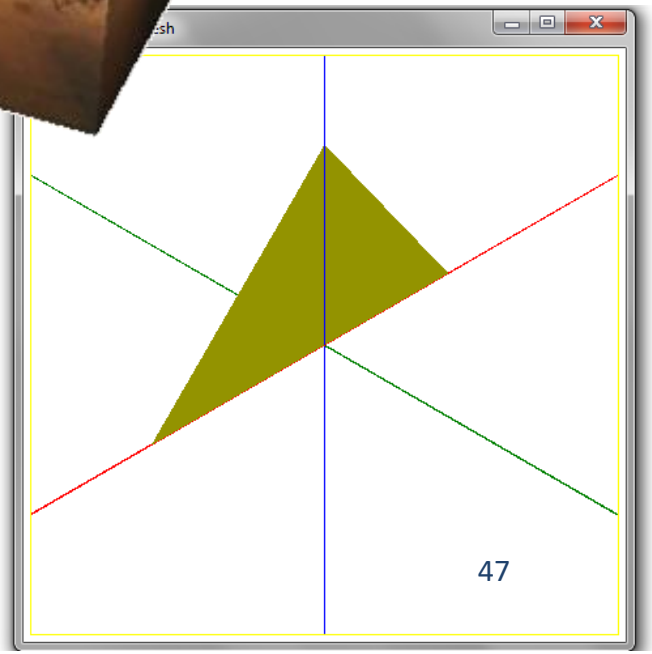


- GeometryModel3D = Mesh + Material (+ Transform)
- Model3D = Lights + GeometryModel3D (of meerdere)
- Model3DGroup = verzameling Model3D-objecten of Model3DGroups

- ViewPort3D = 2D-projectie van de 3D-scene
  (bevat een ModelVisual3D en een camera)

XAML for 3D





Odisee
DE CO-HOGESCHOOL

XAML For 3D triangle

Window

```
<Window x:Class="DrawingIn3D.OneTriangleMesh"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
       Title="OneTriangleMesh" Height="300" Width="300" >

    <Grid Margin="5">
      <Border BorderBrush="Yellow" BorderThickness="1">
```

Viewport

```
        <Viewport3D>
```

Camera

```
          <Viewport3D.Camera>
            <PerspectiveCamera
              Position="-2,2,2"
              LookDirection="2,-2,-2"
              UpDirection="0,1,0"
              />
          </Viewport3D.Camera>
```

```
          <ModelVisual3D>
              <ModelVisual3D.Content>
```

Light

```
                  <DirectionalLight
                    Color="White"
                    Direction="-1,-1,-1" />
              </ModelVisual3D.Content>
          </ModelVisual3D>
```

Figure 1
2D Coordinate System

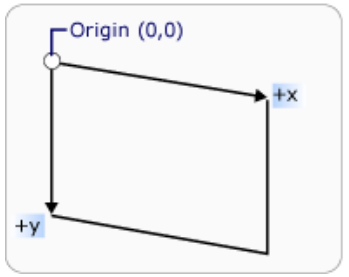


Figure 2
3D Coordinate System

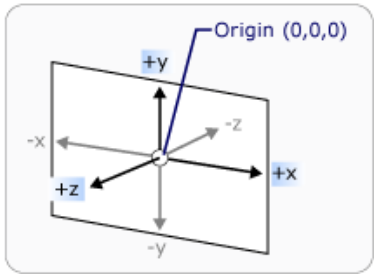
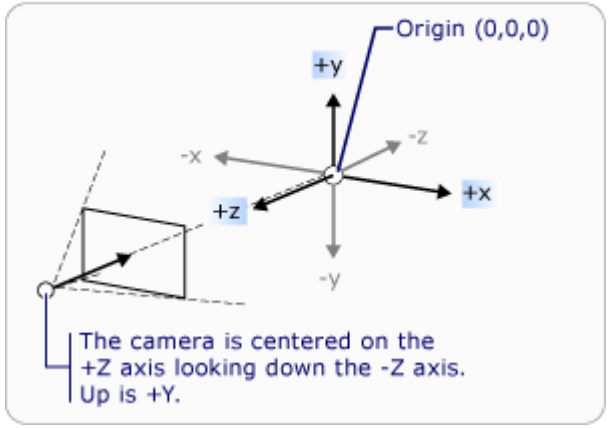


Figure 6
Our camera setup



Odisee
DE CO-HOGESCHOOL

XAML for 3D Triangle

Points

Sequence of points

Material

Comments

```
<ModelVisual3D>
    <ModelVisual3D.Content>
      <GeometryModel3D>
            <GeometryModel3D.Geometry>
                <MeshGeometry3D
                    Positions="-1,0,0 0,1,0 1,0,0"
                    TriangleIndices="0,2,1" />
            </GeometryModel3D.Geometry>


            <GeometryModel3D.Material>
                <DiffuseMaterial Brush="Yellow" />
            </GeometryModel3D.Material>

          <!--  <GeometryModel3D.BackMaterial>
              <DiffuseMaterial Brush="Green" />
            </GeometryModel3D.BackMaterial>-->

          </GeometryModel3D>
        </ModelVisual3D.Content>
    </ModelVisual3D>


    </Viewport3D>
  </Border>
 </Grid>
</Window>
```

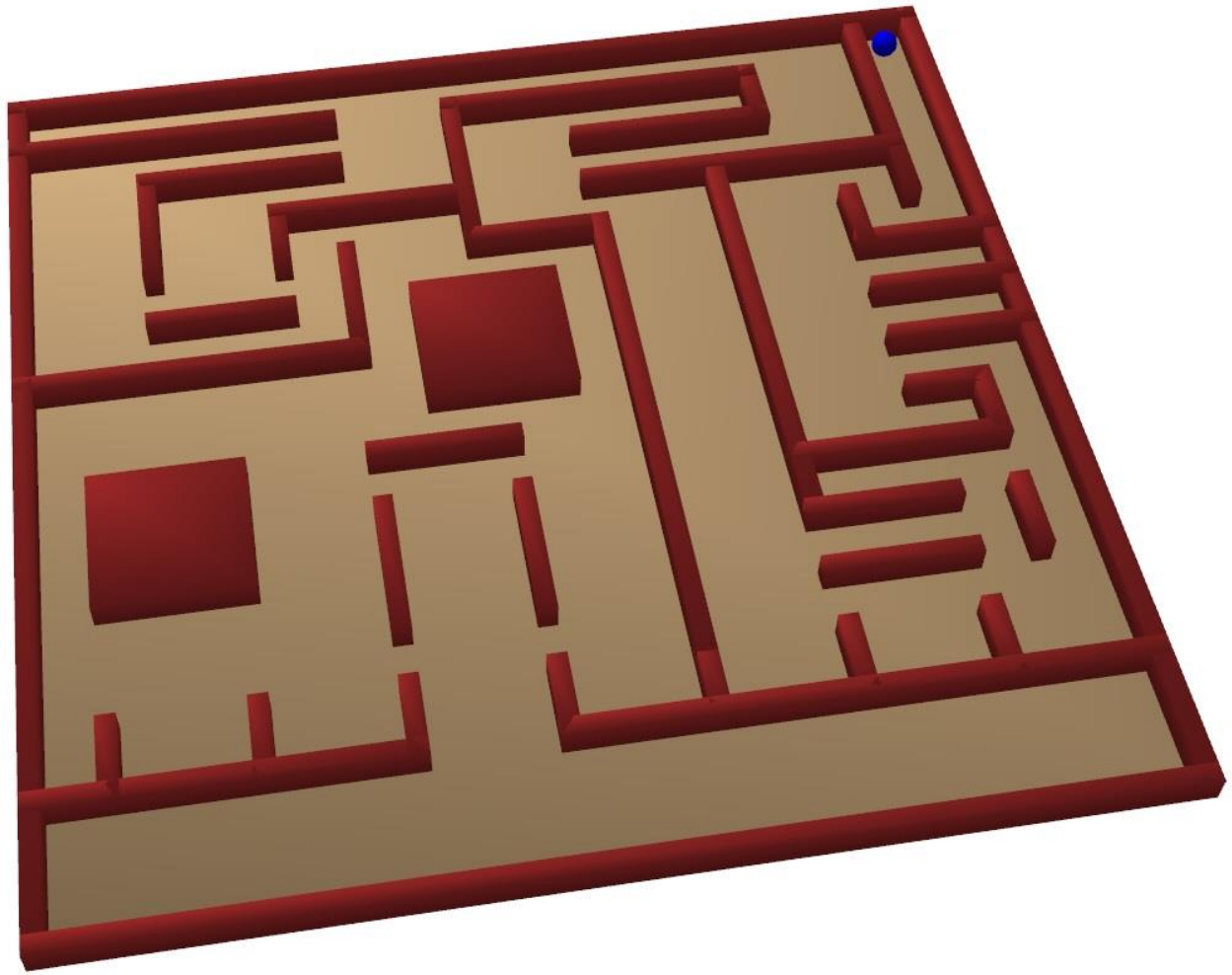
Odisee
DE CO-HOGESCHOOL

6. Opgave

Idee


START
FINISH
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36

Implementeer een 3D doolhof met bewegende bal in WPF



Implementeer een 3D doolhof met bewegende bal in WPF

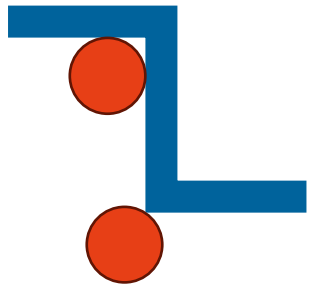
- Ontwerp bord en bal in 3D
- Genereer een doolhof met 1 van de technieken uit vorige opgave
  - Werk verder met je bestaande model
- Het bord kan kantelen door controls in de GUI
- De bal gedraagt zich volgens de wetten van de fysica
  - Het bord kantelt schuin en de bal rolt onder invloed van de zwaartekracht
  - De bal botst met de muren volgens de wetten van de fysica

Genereer een doolhof met 1 van de technieken uit vorige opgave

- Er wordt bij het opstarten een doolhof gegenereerd
- De bestaande algoritmen moeten niet meer aangepast worden
  - Je 2D visualisatie uit opgave 1 blijft ook werken
- Je krijgt een doolhof van het algoritme en bouwt de 3D wereld op
- Als de speler het doolhof heeft ‘opgelost’, genereer je een nieuw doolhof en gaat de simulatie verder
  - Wat je verstaat onder ‘opgelost’ kies je zelf

De bal gedraagt zich volgens de wetten van de fysica

- Bij de start staat het bord perfect horizontaal
- De camera kan recht boven het bord staan of op een andere positie
- De speler kan via de gui het bord kantelen, voorzie grenzen
- Doordat het bord schuin staat, begint de bal naar beneden te rollen
- Als de bal een muur raakt, treedt een realistische elastische botsing op
- Let op met botsingen met meerdere muren



Vereisten

- Werk verder in de bestaande solution, voeg projecten toe
- Code in C# en XAML
- Als 3D-raamwerk maak je gebruik van WPF dat rechtstreeks gebruik maakt van DirectX
  - 3dTools mag
  - Helix Toolkit mag

Documentatie

- readme.md
  - Beschrijving nieuwe structuur
  - Beschrijving aanpassingen aan bestaande code
  - Beschrijving fysica
    - Verwijzing naar wetten uit de theorie
    - Toelichting implementatie van deze wetten
  - Beschrijving 3D model
    - Schematische weergave/diagram
    - Beschrijving opbouw model
  - Bespreking van werking grafische applicatie + screenshot
  - Algemene bedenkingen / reflectie

! Werk verder in hetzelfde bestand
- 1 hoofdstuk per opgave