Code of Conduct

Dit practicum wordt gequoteerd, en het examenreglement is dan ook van toepassing. Soms is er echter wat onduidelijkheid over wat toegestaan is en niet inzake samenwerking bij opdrachten zoals deze.

De oplossing die je voorlegt aan de assistent die je practicum beoordeelt moet volledig het resultaat zijn van het werk dat je gepresteerd hebt. Je mag je werk uiteraard bespreken met andere studenten, in de zin dat je praat over algemene oplossingsmethoden of algoritmen, maar de bespreking mag niet gaan over specifieke code die je aan het schrijven bent. Als je het met anderen over je practicum hebt, mag dit er dus **NOOIT** toe leiden, dat je op om het even welk moment in het bezit bent van een volledige of gedeeltelijke kopie van het opgeloste practicum van anderen, onafhankelijk van of die code nu op papier staat of in elektronische vorm beschikbaar is, en onafhankelijk van wie die code geschreven heeft (mede-studenten, eventueel uit andere studiejaren, volledige buitenstaanders, e.d.). Dit houdt tevens ook in dat er geen enkele geldige reden is om de code van je practicum door te geven aan mede-studenten.

Elke student is verantwoordelijk voor de code en het werk dat hij indient. Indien je Java code verkrijgt op een dubieuze wijze en die in het practicum inbrengt, word je verantwoordelijk geacht. Als tijdens de demonstratie van het practicum de assistent twijfels heeft over het feit of het practicum zelf gemaakt is (bvb. gelijkaardige code met andere studenten), zal de docent worden ingelicht en de quotering in beraad gehouden worden. Een tweede demonstratie zal dan eventueel gehouden worden, in aanwezigheid van een tweede assistent en de docent. Indien dit de twijfels niet wegwerkt, kan er overgegaan worden tot het melden van een onregelmatigheid, zoals voorzien in het examenreglement.

Practicum Methodiek van de Informatica
Solar

25 april 2006

Opmerking : al het extra materiaal (extra bestanden, klassen, packages, documentatie, voorbeeldcode) dat in deze opgave besproken wordt kan je steeds terugvinden op of downloaden van de Toledo-pagina's van dit vak.

1 Inleiding

In dit practicum ga je een systeem ontwikkelen waarmee je een 3D-scène moet visualiseren. Je programma moet toelaten om allerlei geometrische figuren toe te voegen, te wijzigen en te verwijderen uit je scene. Tevens zal je programma op basis van je locatie op aarde en de datum de positie van de zon moeten bepalen en op basis hiervan de eigenschaduw en de slagschaduw van de voorwerpen in je scene moeten weergeven.

In sectie 2 lichten we de opgave meer in detail toe, en geven we de vereisten waaraan je programma moet voldoen. De formaliteiten die je moet volgen om je practicum in te dienen en te verdedigen leggen we uit in sectie 4. Om je een beetje op weg te helpen beschrijven we in 5 de wiskundige formules die je moet gebruiken om de locatie van de zon te bepalen gegeven een tijdstip en locatie. Ook wordt toegelicht hoe je objecten kan roteren, scaleren en transleren in je scene. Tot slot geven we aan hoe je de coördinaten van de schaduw kan berekenen op het grondvlak.

Je kan bij de uitwerking van je programma vertrekken van een bibliotheek van klassen die voor jullie werden gemaakt. Deze bibliotheek bevat zowel een uitgebreide grafische gebruikersinterface (`SolarGui`) als een hulpklasse om 3D vectoren voor te stellen en hiermee te rekenen (`Vector3D`).

2 Visualiseren van een scene

2.1 Voorwerpen in een scene

Een scene, zoals ze in ons practicum beschouwd wordt, bestaat uit een aantal voorwerpen die je zelf moet kunnen ontwerpen. Deze voorwerpen zijn eenvoudi-

Materiaal	*Dichtheid*
Kurk	130
Eikenhout	800
Asfalt	1300
Glas	2600
Aluminium	2700
Steen	2800
Ijzer	7800

Tabel 1: De dichtheid van enkele materialen

ge 3-dimensionale geometrische figuren. We willen dat je programma minimaal volgende voorwerpen ondersteunt:

Balk: beschreven door een hoogte, een breedte en een lengte

Kubus: beschreven door een lengte

Piramide: beschreven door een zijde van het grondvlak en een hoogte

Prisma: beschreven door een hoogte, een breedte en een lengte

Bij de aanmaak van een voorwerp moet je ook specificeren welke kleur het voorwerp heeft en uit welk materiaal je voorwerp bestaat. In de huidige opgave, hebben we van het materiaal enkel de dichtheid nodig, waarvoor je de informatie in tabel 2.1 kan gebruiken.

Van elk voorwerp dat we aanmaken moeten we minimaal kunnen opvragen wat zijn volume en gewicht is, alsook het aantal vlakken dat je voorwerp heeft.

2.2 Locatie van een voorwerp

2.2.1 Initiële locatie

Eens je object is aangemaakt, moet je het op een bepaalde locatie in je scene kunnen toevoegen. Initieel voeg je je voorwerp toe door de coördinaten in te geven van het ankerpunt van het toegevoegde object. Waar dit ankerpunt ligt is een kwestie van afspraak, maar voor de voorwerpen die in dit practicum worden gebruikt veronderstellen we dat dit ankerpunt in het centrum van het ondervlak van het voorwerp ligt.

2.2.2 Geldigheid van de locatie

Je programma moet intelligent genoeg zijn om te na te gaan of de scene die je wil aanmaken fysiek mogelijk is. Je mag dus nooit twee voorwerpen door elkaar laten gaan. Om de ruimtemeetkunde tot een absoluut minimum te beperken mag je als vereenvoudiging stellen dat een voorwerp overlapt met een ander voorwerp als hun kleinst mogelijke omhullende balk (bij elk van de opgegeven figuren is die makkelijk te bepalen) een doorsnede heeft.

2.2.3 Aanpasbaarheid van de locatie

Eens een voorwerp is toegevoegd aan je scene moet je het ook kunnen verplaatsen binnenin deze scene. Om de controle van het overlappen met andere objecten eenvoudig te houden, moet je enkel translaties (verplaatsingen van alle punten van het voorwerp volgens eenzelfde vector) ondersteunen, en is de ondersteuning van rotaties niet gevraagd.

2.3 Informatie over je scene

Naast het visualiseren van je scene, moet je ook een aantal basisgegevens kunnen opvragen. Concreet moet de gebruiker:

1. Het aantal voorwerpen in je scene kunnen opvragen
2. Het gemiddelde gewicht en gemiddelde volume van de voorwerpen in de scene kunnen opvragen
3. Het zwaarste object in de scene kunnen opvragen

2.4 Bekijken van je scene

Het visualiseren van je scene wordt grotendeels aangeboden als bibliotheek met de klasse `SolarGui`. Je zal je scene vlak per vlak aan de bibliotheek moeten doorgeven in de vorm van een punten-array. De bibliotheek zal vervolgens alle vlakken die eraan zijn meegegeven tekenen in 3D. Er wordt ook verwacht dat je de kleur van je vlak en het verlichtingsniveau meegeeft. Het is van essentieel belang dat je de verschillende punten van je vlak in de juiste volgorde in deze array steekt, anders zullen de punten foutief getekend worden op het scherm.

De bibliotheek onthoudt alle vlakken die je er in het verleden aan hebt meegegeven. Wanneer een vlak dus niet meer thuis hoort in je scene, zal je dat vlak expliciet moeten verwijderen. Hiervoor dien je dezelfde punten-array mee te geven die je indertijd hebt gebruikt om het vlak toe te voegen.

Het verlichtingsniveau waarvan sprake in de bovenstaande paragrafen is een parameter die gebruikt wordt om de eigenschaduw van een object weer te geven.

2.4.1 Eigenschaduw en verlichtingsniveau

De eigen schaduw is de schaduw die op het object aanwezig is. Bijvoorbeeld, als de zon schijnt op een paal dan is de ene kant van de paal licht en de andere donker. In dit practicum vereenvoudigen we de eigenschaduw door 10 niveaus van verlichting te gebruiken die je kan meegeven aan de GUI om een vlak mee te tekenen.

Welk verlichtingsniveau een bepaald vlak heeft moet je zelf bepalen. Beschouw hiertoe twee vectoren: $\overrightarrow{n}$ en $\overrightarrow{z}$ waarbij $\overrightarrow{z}$ de richting van de zon is en $\overrightarrow{n}$ de normaalvector op het vlak, met hiertussen een hoek α, uitgedrukt in graden. Je verlichtingsniveau stel je dan in op $\frac{100-\alpha}{10}$, waarbij je dit getal afrondt naar beneden.

Als de hoek tussen je normaal en de richting van de zon dus 46 graden is, dan moet je verlichtingsniveau voor dat vlak ingesteld worden op 5. Hoeken groter dan 90 krijgen dus een verlichtingsniveau van 0.

2.4.2 De slagschaduw

Het tekenen van de slagschaduw van je objecten is volledig analoog aan het tekenen van voorwerpen in je scene. Eens je de coördinaten van de slagschaduw van een voorwerp op het grondvlak hebt gevonden, zal je deze op eenzelfde manier moeten tekenen als de andere vlakken uit je scene. Om de schaduw te kunnen zien op de (standaard zwarte) achtergrond, kiezen we voor een schaduw als basiskleur grijs.

De wiskunde achter het berekenen van deze slagschaduw wordt even gerecapituleerd in sectie 5.

3 Hoe beginnen?

Met dit practicum willen we vooral nagaan hoe goed je object-georiënteerde concepten zoals encapsulatie en overerving begrijpt en hoe goed je deze kan toepassen in een mooi programma. De nadruk zal bij de verbetering dan ook liggen op je ontwerp: welke klassen je hebt, welke functionaliteit je waar steekt en hoe deze klassen met elkaar samenwerken.

Begin dus eerst met een ontwerp van het programma. Hierbij willen we dat je goed nadenkt over de concepten die voorkomen in het programma en die je gaat voorstellen als klassen. Het ontwerp van het programma kan je opstellen door op volgende vragen een antwoord te zoeken: welke reeds bestaande klassen ga je gebruiken (de meegegeven klassen, klassen die reeds beschikbaar zijn in de Java API)? Welke zelf te definiëren klassen en objecten heb je nodig? Welke operaties moeten gedefinieerd worden? Welke attributen moeten worden bijgehouden voor de zelf-gedefinieerde klassen? Werk dit eerst uit op papier voor je begint te programmeren en motiveer je beslissingen!!!

Je implementeert dan vervolgens je programma. Begin eerst met de klassen die je wereld voorstellen. Vervolgens schrijf je je hoofdprogramma dat, gebruik makende van de GUI en je zelf geschreven klassen, je scene tekent. Tot slot probeer je hieraan ook nog de slagschaduw toe te voegen.

4 Wat in te leveren?

Er wordt van jullie verwacht dat je in dit practicum een **werkend** programma schrijft. Jullie moeten de oplossing van je practicum elektronisch indienen **vóór 12u00** op vrijdag 12 mei 2006. Meer instructies daarover komen op de Toledo pagina's. Met oplossing bedoelen we alle `java`-bestanden, waarin je zelfgemaakte klassen geplaatst zijn. De gegeven klassen van de GUI zullen reeds aanwezig zijn bij je verdediging en moet je dus niet opsturen.

Naast de elektronische indiening moet je ook een papieren verslag inleveren bestaande uit:

- **Een voorblad** dat zal worden opgestuurd bij het elektronisch indienen van het practicum.
- **Een object-schema** van je programma. Hierin moeten zeker al je eigen klassen terug te vinden zijn alsook hun voornaamste attributen. Dit object-schema moet een goed overzicht geven van de opbouw van je objecten en de relaties die er bestaan tussen deze objecten. Extra informatie met het maken van een object-schema kan je terugvinden op Toledo.
- **Een klassen-schema** van je programma die de statische structuur van je oplossing beschrijft.
- **Een overzicht van je ontwerpbeslissingen**. Deze ontwerpbeslissingen zijn voornamelijk een opsomming van:
  - de niet zelf geschreven klassen die je gebruikt, en waarvoor je ze gebruikt.
  - de zelf geschreven klassen, en wat objecten van deze klassen voorstellen in jouw programma

  Dit ontwerp-verslag moet kort en bondig zijn. Eén bladzijde kan meer dan voldoende zijn.

Dit papieren verslag moet gedeponeerd worden **vóór 17u00** op vrijdag 12 mei in een doos die we zullen plaatsen voor het bureau van Pieter-Jan Drouillon (Monitoraat ingenieurswetenschappen 200A 00.28)

Het practicum mag je oplossen in groepjes van 4 personen. Toch wordt er verwacht dat iedereen een afzonderlijk papieren verslag indient. Dat verslag mag voor alle leden van een zelfde groepje identiek zijn, op het voorblad na. Het practicum maakt deel uit van je examen. Het is **niet** toegelaten om over groepgrenzen heen samen te werken.

4.1 Mondelinge verdediging

Kort na de indiendatum moet je je werk mondeling verdedigen. Tijdens jouw mondelinge bespreking zullen de opgestuurde bestanden beschikbaar staan. Bij het demonstreren van je practicum moet je dan eerst een BlueJ project aanmaken waarin je deze bestanden plaatst. **Zorg er dus voor dat je voldoende goed met BlueJ en de Windows Verkenner (Explorer) overweg kan!**

Gedurende een kwartier zal de assistent je ondervragen over je practicum. Deze ondervraging gebeurt analoog aan degene van het proefpracticum in groep van 4 waarbij je wel individueel wordt aangesproken. Deze verdediging duurt 30 minuten per groepje van 4 personen en staat op 8 van de 20 punten van het examen voor het vak Methodiek van de Informatica. Het tijdschema van deze bespreking (wanneer wie waar moet zijn) zal via toledo bekend gemaakt worden.

4.2 Je werk zal beoordeeld worden op...

De beoordeling die je zal krijgen zal ondermeer gebaseerd zijn op het juiste gebruik van OO-concepten, het gebruik van klassen en objecten, en de manier waarop je je programma indeelt in klassen. Het is belangrijk dat je programma makkelijk uitbreidbaar is (m.a.w. als we iets zouden toevoegen aan de opgave zou je ontwerp makkelijk aanpasbaar moeten zijn zodat deze uitbreiding makkelijk kan worden geïmplementeerd).

Daarnaast word je ook beoordeeld op de uitleg die je verschaft bij het demonstreren van je programma. Tijdens de demonstratie zullen we je ook extra vragen stellen, die te maken hebben met de werking en de uitbreidbaarheid van je programma. Het mondelinge aspect speelt dus een belangrijke rol!

Vermits dit practicum deel uitmaakt van het examen van dit vak wordt er geen feedback gegeven tijdens de verdediging. De mondelinge verdediging is verplicht.

4.3 Per groep of individueel?

Het wordt sterk aanbevolen om het practicum in groep (max. 4 personen) op te lossen. Ook de ondervragen gebeurt in groep. Toch zullen vragen rechtstreeks aan alle groepsleden worden gesteld. Je moet dus het volledige practicum begrijpen, niet enkel jouw deel. Bovendien mag er **NIET** worden samengewerkt over de groepsgrenzen heen!!!

Zelfs al maak je het practicum in groep, toch wordt er **één verslag per persoon** verwacht. De inhoud van die verslagen mag dan natuurlijk identiek zijn (wel met een verschillend voorblad).

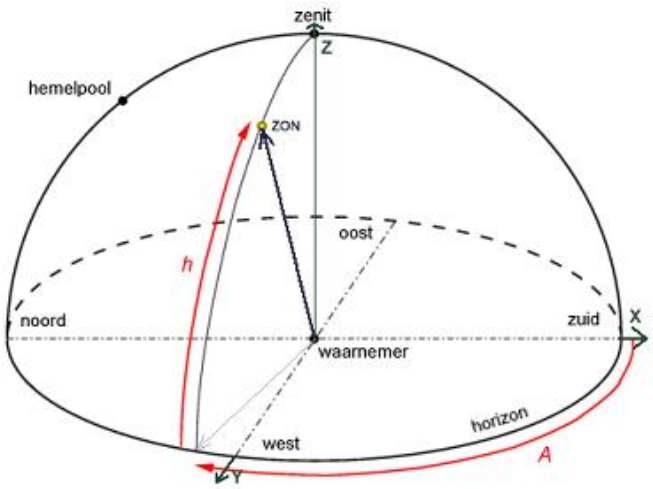
4.4 Wat mag (niet) gebruikt worden?

In dit practicum mag je gebruik maken van de standaard `jdk`-klassen, van de klassen die we ter beschikking stellen (zie ook 6) en natuurlijk van de klassen die je zelf geschreven hebt. **Het gebruik van andere klassen, packages of bibliotheken is niet toegelaten.**

5 Wat achtergrond-wiskunde

Het eerste probleem waarmee je geconfronteerd wordt is het bepalen van de locatie van de zon gegeven je locatie en het tijdstip. Hiervoor worden een aantal formules gebruikt uit de astronomie. Hoewel je natuurlijk deze formules zal moeten implementeren en verwerken in je programma, is het niet vereist voor dit practicum dat je de achtergrond en de afleiding van deze formules kent. Er zullen geen vragen gesteld worden over het wiskundige gedeelte op je verdediging.

Figuur 1: Hoeken van de zon



5.1 Een beetje astronomie

De positie van de zon wordt bepaald door twee hoeken: de azimut en de zonnehoogte, die worden weergegeven op figuur 1. Beide hoeken zijn afhankelijk van het tijdstip en de locatie waarop men zich bevindt. De gegevens waarvan we vertrekken zijn de volgende:

- breedtegraad: breedtegraad van je positie;
- maand: de huidige maand;
- dag: de huidige dag;
- uur en minuten: het huidig tijdstip.

Alle hoeken worden uitgedrukt in graden en we kiezen ons assenstelsel zo dat de X as samenvalt met de Noord-Zuid richting, en de Y-as met de Oost-West richting.

5.1.1 Twee hulpbegrippen: declinatie en tijdshoek

Declinatie

De declinatie is de hoek van het vlak van de evenaar en de verbindingslijn tussen aarde en zon. Deze is afhankelijk van de datum. Op 21/3 (begin van de lente) bedraagt de declinatie 0 graden, op 21/6 bedraagt ze 23.45 graden, op 21/9 terug 0 graden en op 21/12 bedraagt ze -23.45 graden. De declinatie maakt voorts

een sinusoïdale beweging tussenin. We kunnen dan ook makkelijk schrijven dat:

$$declinatie = 23.45 \sin\left(\frac{2\pi * dagenVanafBeginLente}{TotaalAantalDagenInJaar}\right) \tag{1}$$

Je mag uitgaan van een standaardjaar, dus 365 dagen in een jaar (dus geen schrikkeljaar).

Tijdshoek

Dit is een fictieve hoek die het tijdsverschil aangeeft tussen het huidige uur en 12 uur 's middags. Een uur tijdsverschil komt dus overeen met 15 graden.

$$tijdshoek = 15\left[\left(uur + \frac{minuten}{minutenPerUur}\right) - 12\right] \tag{2}$$

5.1.2 Zonnehoogte

De zonnehoogte komt overeen met hoek h op figuur 1, en wordt berekend met volgende uitdrukking:

$$Zonnehoogte = \arcsin(a + b) \tag{3}$$

Hierin zijn de constanten a en b gelijk aan:

$$a = \sin(breedtegraad) * \sin(declinatie) \tag{4}$$

$$b = \cos(breedtegraad) * \cos(declinatie) * \cos(tijdshoek) \tag{5}$$

5.1.3 Azimut

De azimut wordt op figuur 1 voorgesteld door de hoek A. Hij wordt als volgt berekend:

$$azimut = sign(tijdshoek) * \arccos(a) \tag{6}$$

De constante a is gelijk aan:

$$a = \frac{\sin(zonnehoogte)\sin(breedtegraad) - \sin(declinatie)}{\cos(zonnehoogte)\cos(breedtegraad)} \tag{7}$$

Deze formule leidt echter voor een aantal specifieke gevallen tot een incorrect resultaat. Voor die gevallen definiëren we apart:

(i) Als het 12u00 is, dan is de $Azimut = 0°$

(ii) Als de breedtegraad kleiner is dan de declinatie dan is: $Azimut = 180°$

(iii) Als de zonnehoogte 90° is, dan is: $Azimut = 0$

(iv) Als de breedtegraad 90° is, dan is: $Azimut = tijdshoek$

(v) Als de breedtegraad $-90°$ is, dan is: $Azimut = 180° - tijdshoek$

5.1.4 Voorbeeldje

Als je op 25 september om 10u45 op de 53ste breedtegraad staat, dan is de zonnehoogte 32.585474 graden en de azimut -22.408326 graden.

5.2 Rekenen met coordinaten

De positie van de zon is nu gekend op basis van twee hoeken. Om deze twee hoeken om te zetten naar coordinaten zal je de eenheidsvector [1,0,0] moeten roteren over een hoek h rond de Y-as, gevolgd door een rotatie van hoek A rond de Z-as. Vervolgens kan je eventueel deze vector nog scaleren. Roteren van vectoren rond een as doe je door vermenigvuldiging met een zogenaamde rotatiematrix. Voor rotatie over een hoek α rond de Z-as ziet deze er uit als volgt:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{8}$$

De matrices voor een rotatie rond de X- of Y-as zijn volledig analoog. In je practicum kan je echter gebruik maken van de klasse `Vector3D` voor het uitrekenen van deze transformaties.

5.2.1 Voorbeeldje

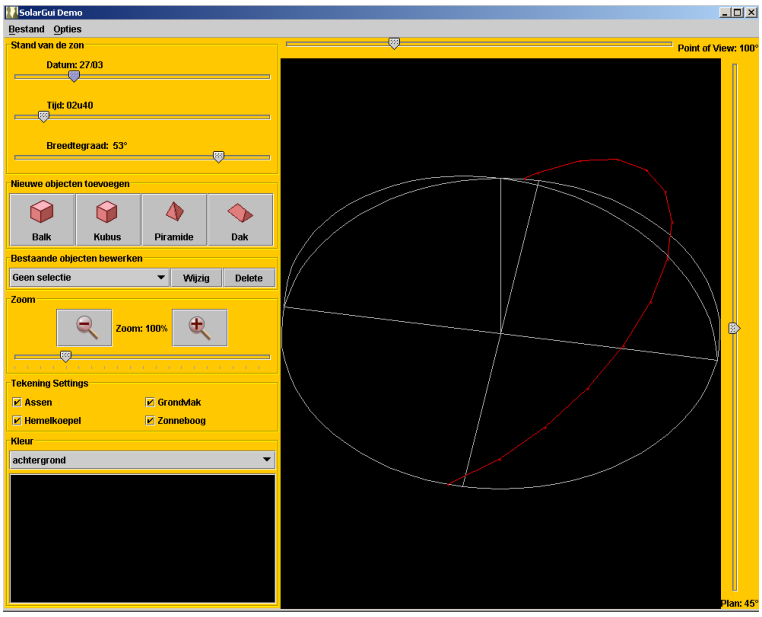
Vertrekkend van de gegevens uit 5.1.4 en je wilt de vector(25,0,0) roteren over de zonnehoogte (Y-as) en azimut (Z-as) dan kom je uit op vector [19.47,-8.02,13.46].

5.3 Schaduwberekenen

Gegeven de locatie van de zon, en de de coordinaten van de hoekpunten van een voorwerp in je scene, kan je nu makkelijk de coordinaten van de schaduw op het grondvlak berekenen. Hiertoe moet je een denkbeeldige rechte trekken van de zon door het hoekpunt, en het snijpunt van deze rechte met het grondvlak (Z = 0) berekenen. Dit doe je dan voor elk hoekpunt (x_i, y_i, z_i). De onbekende r_i kan je uit het stelsel halen.

$$z = 0 \tag{9}$$
$$x = x_i - r_i \cos(zonnehoogte)\cos(azimut) \tag{10}$$
$$y = y_i - r_i \cos(zonnehoogte)\sin(azimut) \tag{11}$$
$$z = z_i - r_i \sin(zonnehoogte) \tag{12}$$

Los dit stelsel op voor alle hoekpunten van de kubus, en voilà: je kent de hoekpunten van de schaduw.



Figuur 2: De GUI

6 Gegegeven Bibliotheken

Om de hoeveelheid werk van het practicum deels onder controle te houden geven we jullie een bibliotheek mee met daarin voorgeprogrammeerde functionaliteit. Deze functionaliteit bestaat uit twee delen: (i) de gebruikersinterface en (ii) een klasse om te rekenen met 3D coördinaten. We verwachten dat jullie in staat zijn om documentatie die geleverd is bij deze klasse te lezen en te interpreteren. We gaan dan in deze opgave ook niet elke methode van de de grafische interface overlopen.

6.1 De Gui: `SolarGui`

De grafische bibliotheek die ter beschikking wordt gesteld zal de vlakken die jij eraan meegeeft tekenen in een 3D omgeving. De Gui heeft een boel toeters en bellen die je vanuit je applicatie niet moet (kan) aanspreken of wijzigen. Zo kan je het standpunt van de waarnemer wijzigen via de *point-of-view* en *plan* sliderbars, kan je inzoomen en uitzoomen op je scene en kan je een hemelboog laten tekenen op basis van de datum en locatie die je in de GUI aangeeft.

In tegenstelling tot wat je misschien kan vermoeden wanneer je naar de Hemelboog kijkt, is de klasse `SolarGui` zuiver grafisch qua opzet: het is niet mogelijk om uit de gui informatie te halen omtrent de stand van de zon om deze informatie in je applicatie te gebruiken.

6.1.1 Herbruikbare Elementen

Naast een aantal GUI-only elementen (zoals de eerder vermelde sliders van point-of-view en plane, maar ook de instellingen van de kleurtjes die door de gui gebruikt worden), zijn er een aantal elementen uit de GUI die je kan opvragen:

4 Object-Knoppen: De 4 knoppen met een balkje, kubus, piramide of dakje op doen standaard niets. Je kan de `JButton` objecten die overeenkomen met deze knoppen wel opvragen aan de Gui. Om je eigen code te laten uitvoeren wanneer op een van deze knoppen wordt geklikt moet je zelf een `ActionListener` schrijven en deze aan de desbetreffende knop toevoegen.

Een pull-down controlbox: Net onder de 4 knoppen zie je een (initieel lege) combobox. Je kan hier zelf `String`-objecten aan toevoegen en op elk ogenblik het geselecteerde element opvragen.

Een `Wijzig` en `Delete` knop: Analoog aan de 4 object knoppen kan je ook eigen functionaliteit achter deze twee knoppen steken. Noteer dat je de functionaliteit van deze knoppen ook afhankelijk kan maken van de combobox ernaast door je code de toestand van deze combobox op te laten vragen.

De menu-balk: Je kan het `JMenuBar` object opvragen dat de menubalk van de gui voorstelt. Via dit object kan je zelf sub-menus toevoegen.

Drie Sliders: Je kan ook de datum-, tijd- en locatie-sliders opvragen bovenaan links in de gui. Hiermee kan je nodige gegevens verkrijgen om de zonlocatie te berekenen. Net zoals aan een knop, kan je ook aan een `JSlider` zelf gedefineerde acties hangen via een `ChangeListener`.

6.1.2 Opvangen van GUI-events: ActionListeners

Je kan nu wel een aantal objecten van de gui opvragen, maar wat moet je daar nu mee? De manier waarop je in Java luistert naar gebeurtenissen van de gui (zoals het geklikt worden op een knop, of het verschuiven van een slider), zijn ActionListeners. Deze volgen het call-back principe: *don't call us, we'll call you.*

Praktisch gaat het als volgt in zijn werk:

1. Je schrijft een klasse die een gegeven interface (ActionListener, ChangeListener, ...) implementeert. Typisch bevat deze interface slechts 1 methode waarin je schrijft hetgeen je wil dat wordt uitgevoerd wanneer de gebeurtenis waarop je je zal registreren voorvalt.

2. Je registreert een object van deze klasse bij het gui-element in kwestie.

3. Wanneer nu de gebeurtenis waarop het object is ingeschreven voorvalt, zal het gui-element de methode die in de actionlistener interface is gespecificeerd uitvoeren op jouw object.

Het voorbeeldprogramma dat bij de bibliotheek is meegeleverd illustreert dat principe. Zo implementeert de klasse `VoegVlakToeActie` de interface `ActionListener` en zijn bijhorende methode, void actionPerformed(ActionEvent e). Vervolgens wordt een object van de klasse `VoegVlakToeActie` geregistreerd bij een knop van de gui:

```
VoegVlakToeActie vvta =
      new VoegVlakToeActie(gui, vlak, normaal,
                           vlak2, normaal2);

gui.getBalkKnop().addActionListener(vvta);
```

De situatie is analoog bij een slider die van positie verandert. Hier zal je echter de interface `ChangeListener` moeten implementeren. Wanneer we telkens de datum-slider van positie verandert, zijn waarde willen afdrukken ziet deze ChangeListener eruit als volgt:

```
public class SomeListener implements ChangeListener
{
 public void stateChanged(ChangeEvent e) {
   int sliderValue = ((JSlider) e.getSource()).getValue();
   System.out.println("De waarde is: " + sliderValue);
 }
}
```

Hoewel de naam van de interface en methode (void stateChanged(ChangeEvent e)) anders zijn is het principe volledig analoog.

Je kan acties koppelen aan alle gui-elementen die je kan opvragen. De gui-layout ligt vast en het is dus niet mogelijk om extra knoppen toe te voegen. Als je acties hebt waar je geen knop voor vindt en die je toch aan de gui wil koppelen dan zal je deze dienen toe te voegen aan de menustructuur.

We verwijzen verder naar de Javadoc van de gui-elementen `JButton`, `JComboBox`, `JMenuBar`, `JMenuItem` en `JSlider` en de naar de documentatie van de listeners `ChangeListener` en `ActionListener` voor meer informatie.

6.1.3 Een kant-en-klare dialoog

De klasse `SolarGui` biedt een methode aan die informatie vraagt aan de gebruiker, en dat vervolgens in de vorm van een String array aan je teruggeeft via de methode `vraagInfo`.

```
public String[] vraagInfo(String vraag, String title,
                  String[] waarden, String[] defaults)
```

Deze methode is bedoeld om makkelijk ineens een aantal eigenschappen (bv: hoogte, breedte en diepte) aan de gebruiker te vragen. Voor elke eigenschap die je wil steek je een `String` met beschrijving in de array waarden. De co-geïndexeerde array `defaults` bevat de standaardwaarden van je eigenschappen

die je opvraagt. Als de gebruiker zijn keuze bevestigt, geeft de methode alle antwoorden in volgorde in een String-array terug.

Het spreekt voor zich dat het gebruik van de methode `vraagInfo` niet verplicht is. Het is toegestaan om eigen dialoogvensters te ontwikkelen en deze te koppelen aan de Gui. Je krijgt hier echter geen extra punten voor.

6.1.4 Tekenen van vlakken

De `SolarGui` voorziet drie methodes die hetgene hij tekent beïnvloeden:

```
public void voegVlakToe(Vector3D[] punten,
      int verlichtingsniveau, Color kleur,
      Vector3D normaal);
public void veranderVlak(Vector3D[] punten,
      int verlichtingsniveau, Color kleur,
      Vector3D normaal);
public void verwijderVlak(Vector3D[] punten);
```

De eerste methode voegt een nieuw vlak toe aan de bibliotheek. Deze zal aanwezig blijven in de getekende scene tot je het vlak expliciet verwijdert. In de gui wordt een vlak enkel getekend als je naar zijn voorkant kijkt. De achterkant van een vlak wordt dus niet getekend. Om aan te tonen welke kant van je vlak de voorkant is en welke kant de achterkant moet je ook de normaalvector op je vlak meegeven. Als voorkant wordt dan die kant genomen waar je naar kijkt als de normaalvector naar je toewijst.

Eens een vlak is toegevoegd kan je zijn eigenschappen (bv verlichtingsniveau) later nog wijzigen met de methode `veranderVlak`. Je kan het vlak terug verwijderen met de laatste methode. Noteer dat de array die je gebruikt om initieel je vlak toe te voegen aan de Gui tevens door de gui gebruikt wordt als identificatie. Als je dus een vlak wil wijzigen of verwijderen, dien je **dezelfde** puntenarray te gebruiken dan degene die gebruikt is om het vlak in de eerste plaats toe te voegen.

6.2 Rekenen met 3D-Vectoren: `Vector3D`

De klasse `Vector3D` bevat een boel basisfunctionaliteit om te rekenen met 3D vectoren. De methodes die aangeboden worden spreken voor zich, en we verwijzen dan ook naar de javadoc voor meer informatie.

Heel Veel Succes!
Het MI-Team