

ITE 1605 - Datamaskingrafikk

Rapport til karaktergivende oppgave

501629 - Knut Lucas Andersen



2012

Innhold

Innledning.....	2
Oppgaveteksten	3
Forklaring til planlagt program.....	4
Kravspesifikasjon	5
Tidsplan	6
Avsluttende kommentar	6
Vedlegg 1 - Kilder og referanser	7
Vedlegg 2 - Kildekode	9
StartDemo	9
FlaggStruktur	11
LydAvspilling.....	12
PartikkelInformasjon og Partikkeleffekt.....	13
SpriteBehandling	17
Modellbehandling	21
QuaternionKamera.....	28
XNAUtils.....	39

Innledning

Dette er et dokument som inneholder en utdypende forklaring og kravspesifikasjon til et tenkt program som skal utvikles i faget ITE1605 - Datamaskingrafikk. Dokumentet inneholder selve oppgaveteksten, en forklaring til programmet (*hvordan jeg har tenkt at programmet skal se ut og fungere*), etterfulgt av en kravspesifikasjon som inneholder de direkte kravene til hva programmet skal gjøres.

Programmet teller 50% av endelig karakter i dette faget, og programmet baseres og bygges opp under forelesning og øvinger hittil gitt i faget, og er da en praktisk måte å gjengi vårt kunnskapsnivå. Resterende karaktergivning baseres på en eTest som baseres på det teoretiske kunnskapsnivået.

Til slutt er det verdt å nevne at jeg skal utføre denne oppgaven og programmeringen alene, da jeg føler med mitt kunnskapsnivå (*og tidsplan i forhold til andre fag*) at det for meg og andre vil bli for stor belastning dersom jeg skulle gjort dette på et gruppenivå.

Jeg har nå lagt til avsluttende kapitler med avsluttende kommentar, kildereferanser og kildekode på slutten av denne rapporten. Dette for å gi en korrekt kildehenvisning, og hvis ønskelig at kildekoden skal være tilgjengelig på mer lesbar form.

Oppgaveteksten

Oppgaven går ut på å utvikle et C#/XNA-program som inneholder en selvvalgt animasjon/demo/spill med en bevegelig og styrbar figur/modell.

Eksempler på animasjon/spill/demo:

- Simulering av en heis i en bygning. Her kan man tenke seg at man ser heissjakta (rammeverket) og «heiskabinen» som beveger seg opp og ned i forhold til hvilke knapper som trykkes. Her kan man også tenke seg simulering av personer som kjører heis.
- En fabrikkhall med transportbånd og skyvearmer som sorterer "bokser" i ulike størrelser i ulike traller/vogner. Når trallene er fulle kan disse kjøres til pakkemaskin e.l.
- Vektstang. Slipp en ting (boks) på en vektstang for å kaste en ting som ligger på den andre siden av vektstangen. Velg ulike vekter.
- En figur/ kjøretøy som skal kunne styres i en "verden" og samle inn "ting" (bokser) som gir poeng. Elementer som: Farlige soner, soner med fiender som skyter på deg o.l. Tid eller poeng.
- En tenkt byggeplass med heisekran som bygger og stabler (ting/bokser e.l.). Styrbar heisekran.
- Styrbar kanon som skyter på fallende gjenstander. Gravitasjon og kulebaner er aktuelle elementer. Tid og poeng.

Løsninga skal inneholde følgende elementer:

- En egenkonstruert sammensatt 3D-modell. Eks. en robotarm, styrbar heisekran, figur, kjøretøy eller lignende.
- Bruk av minst en importert 3D-modell generert i eksternt verktøy.
- En av disse skal kunne styres via tastaturet.
- Terreng
- Skybox
- Teksturer
- Bruk av egendefinert shader med grunnleggende lysberegning.
- 2D Sprites (skilt, HUD og lignende)
- Kollisjonsdeteksjon
- Minst en partikkeleffekt

Oppgaven teller 50% av karakteren i faget.

Vurderingskriterier:

- Design & generell objektorientering.
- Funksjonalitet og brukervennlighet.
- Kodedokumentasjon.
- Generell ryddighet i koden (bl.a. navngiving av variabler og klasser). Bruk kodestandard.
- Hvor mye av teori og teknikker vi har vært gjennom som brukes i løsninga.
- Det er viktig at koden er egenutviklet og ikke "lånt" fra medstudenter eller andre. I motsatt fall vil dette oppfattes som fusk med de konsekvenser det har (ref. eksamensforskriften: <http://www.hin.no/index.php?ID=2303>).
- I tvilstilfeller vil du/dere kunne bli bedt om å redegjøre for besvarelse muntlig.
- Det forventes en relativt fyldig besvarelse da den teller 50% av total karakteren i faget.

Forklaring til planlagt program

Programmet skal som nevnt i innledning (*og oppgavetekst*) utvikles i C# - XNA. Det jeg har tenkt å forsøke å utvikle er et bilspill/bildemo hvor bruker kontrollerer en bil som kjøres på en vei. Bilen vil da være kjøretøyet som bruker kontrollerer og det er ønskelig at spiller kan velge mellom å bruke piltaster eller WASD for å bevege kjøretøyet.

Partikkeleffekter vil da gjengis ved f.eks. støv på veien, og 2D Sprites vil bli brukt til f.eks. trafikklys og gjengivelse av bilens hastighet. Kjøretøyet som vil bli brukt er en bil som ble utviklet i faget ITE 1606 - 3D Modellering (*verktøyet brukt var 3DS Max*).

Poeng vil kunne gjengis ved at bruker ikke krasjer (*hindringer i veien*) eller muligens treffer/kjører over objekter (*små flagg el.l.*). Dette vil og dekke opp under kravet til kollisjonsdeteksjon, nemlig om kjøretøyet treffer et objekt som øker/senker poengsum.

Jeg er litt usikker på hvordan oppbyggingen av terrenget (*og da tilhørende skybox*) skal se ut, men jeg ser for følgende alternativer (*eller kombinasjon av disse*);

- 1: Bilen kjører inne på en tribune/stadion
- 2: Bilen kjører på en grusvei, i en ørken ell. l. (*hovedsaklig "offroad"*)
- 3: Et form for dragrace hvor bruker kjører mot AI/annen spiller

Selv om det siste alternativet nok har for høy vanskelighetsgrad for meg, så ville jeg bare dekke opp de mulighetene jeg kom på.

Kravspesifikasjon

Krav til programmet

- Programmet skal utvikles i C# - XNA
- Programmet skal inneholde minst en importert 3D-modell fra eksternt verktøy
- Programmet skal ha minst ett kjøretøy som bruker skal kunne styre (*piltaster eller WASD*)
- Programmet skal inneholde 2D Sprites
- Programmet skal inneholde minst en partikkeleffekt
- Programmet skal inneholde minst en kollisjonsdeteksjon
- Programmet skal ha i bruk minst en egendefinert shader med grunnleggende lysberegning
- Programmet skal ha hindringer (*objekter i veien*)
- Programmet skal inneholde lyder for å gjøre hendelser mer realistisk

Krav til kjøretøyet og objekter

- Kjøretøyet skal starte med å kjøre på en vei
- Kjøretøyet skal (*hvis mulig*) kunne kjøre av veien, men kjøretøyet skal ikke "settes fast" (*dersom man kjører av veien, skal man kunne komme seg tilbake på den uten å måtte restarte*)
- Kjøretøyet vil være basert på modell utviklet i 3DS MAX
- Kjøretøyet vil ikke få bulker eller eksplodere ved for mange krasj
- Det skal være objekter gir spiller informasjon (*skilter, HUD*) og objekter som gir spiller øker/reduserer poengsum

Tidsplan

Oppgave/plan	Dato
Oppstart/oppbygging av prosjekt	10.11.2012
Utvikling av nødvendige objekter (3DS MAX)	11.11.2012
Utvikling av HUD og Sprites (<i>grafisk</i>)	14.11.2012
Laste inn ferdiglagde objekter i prosjektet	15.11.2012
Hovedoppstart av selve programmeringen	16.11.2012
Fullføring av resterende og endelig debugging/testing	26. - 29.11.2012
Innlevering av oppgaven	30.11.2012 kl. 23:55

NB! Dette er kun tenkte datoer, da det på gjeldende tidspunkt er ukjent hvor stort størrelsesomfanget for hver enkelt deloppgave blir.

Avsluttende kommentar

Jeg har her tenkt å utdype kort noen tanker/forklaringer til programmet. Når det kommer til størrelsesforholdet mellom bilen og stadion/tribunen, så er dette et problem tilknyttet selve utviklingen av modellene. Disse er lagd med forskjellige størrelsesforhold, og selv med "krymping" i 3DS MAX så er fortsatt bilen liten i forhold. Siden det ville blitt for mye jobb å redigere tribunen, valgte jeg å la den ha dette, om enn skjeve størrelsesforholdet.

Videre så bygde jeg opp prosjektet med den tankegangen at alt skulle ligge i biblioteket, og at alt som lå i Game - klassen (*her: StartDemo*) skulle kun kalle på disse. Derfor lagde jeg alle metoder i biblioteket foruten visse konstruktører internal.

Jeg valgte også å gjøre visse metoder static. Grunnen til dette var at jeg syntes det ble for dumt å lage objekter av en klasse, når klassen med dette objektet kun kalte på en metode. I tillegg så er statiske metoder raskere enn vanlige metoder.

Etter dette følger Vedlegg1 som inneholder kilder og referanser som jeg har brukt under utvikling, og Vedlegg 2 som inneholder kildekoden til alle klasser og strukturer brukt i dette prosjektet.

Vedlegg 1 - Kilder og referanser

Teksturer brukt under modellering av tribune

Kilde	Gjelder	Funnet på
3DS MAX	Tekstur brukt på rampe, trapp og "tunnel" mellom innside/utside bilinngang	Ligger i mappen: 3DS MAX/sceneassets (dette er ikke synlig i bildemoen, men tas med for referansens skyld)
Autodesk 3DS MAX 2010, ISBN (978) 0240811949	Tekstur Himmel	Vedlagt prosjektfiler i boka; mappen project3
Internett	Gresset i stadion	http://s364.photobucket.com/albums/oo85/jpfphoto/?action=view&current=GrassTexture.png&sort=ascending
Internett	Tribuneseter på stadion	http://fotball.aftenposten.no/multimedia/archive/00143/DnbNorArena01-2_143369o.jpg
Internett	Trevirke/Planker	http://agf81.deviantart.com/#/d5fz4z1
Internett	Gulv garasje og bensinstasjon	http://agf81.deviantart.com/art/Seamless-Floor-Special-322523166?q=gallery%3Aagf81%2F31629552&qo=4
Internett	Tekstur vegg på tribunen	http://agf81.deviantart.com/art/Wall-Texture-35-254902111?q=gallery%3Aagf81%2F31629392&qo=20
Internett	Tekstur på veien inne i tribunen	http://agf81.deviantart.com/art/Seamless-Texture-5-160257878?q=gallery%3Aagf81%2F31629552&qo=40
Internett	Tekstur på taket til garasje og bensinstasjon	http://agf81.deviantart.com/art/Roof-Texture-3-317658025?q=gallery%3Aagf81%2F31629337&qo=8
Internett	Ventil på tak garasje	http://agf81.deviantart.com/art/Misc-Texture-60-305272464?q=gallery%3Aagf81%2F31629337&qo=19
Internett	Dør på garasje og inngang/utgang tribune	http://agf81.deviantart.com/art/Door-Texture-2-198834874?q=gallery%3Aagf81%2F31629419&qo=40

Bakgrunnsmusikk og lyder

Kilde	Gjelder	Funnet på
Internett	Skid.mp3	http://www.flashkit.com/soundfx/Cartoon/Skids/skid-Texavery-8946/index.php
Internett	Partybie.mp3	http://www.flashkit.com/loops/Ambient/Electronica/partybie-partybie-10029/index.php
Internett	DrDestru.mp3	http://www.flashkit.com/loops/Ambient/Electronica/DrDestru-Lexicon-9661/index.php
Internett	Atmos.mp3	http://www.flashkit.com/loops/Ambient/Electronica/Atmos-ho-Solid-9089/index.php

Programmeringskilder og kildekode

Kilde	Gjelder	Funnet på
XNA 3.0 Game Programming Recipes - Riemer Grootjans, ISBN 978-1-4302-1855-5	XNAUtils Quaternion Kollisjonsdeteksjon	s. 102 s. 50 - 57 s. 321, s. 325
XNA Game Studio 4.0 Programming, Tom Miller, Dean Johnson	Tegning av sprite (2D)	s. 22
Forelesningsnotater fra Its Learning; "Del 14 - Kollisjonsdeteksjon.pdf"	Kollisjonsdeteksjon	s. 4
Internett	Skybox/Skysphere	http://msdn.microsoft.com/en-us/library/bb464016(v=xnagamestudio.40).aspx
Internett	Zip fil med kildekode og eksempler for Skybox	http://xbox.create.msdn.com/assets/cms/docs/GameStudio3/SkySphere_Sample.zip
Internett	Partikkeleffekter	http://www.riemers.net/eng/Tutorials/XNA/Csharp/Series2D/Particles.php
Internett	Tilbakestilling av variabler ved bruk av SpriteBatch/Font	http://blogs.msdn.com/b/shawnhar/archive/2010/06/18/spritebatch-and-renderstates-in-xna-game-studio-4-0.aspx

Modeller som er brukt

Er alle laget av meg, men deler av tekstur er tatt fra eksterne kilder (s.7). Jeg har også laget bildet/Sprite som brukes som visning av aktivt gir.

Vedlegg 2 - Kildekode

StartDemo

```
/// <summary>
/// Starter avspilling av bildemoen
/// </summary>
public class StartDemo : Microsoft.Xna.Framework.Game {
    #region VARIABLER
    //pre-deklarererte objekter
    private GraphicsDeviceManager graphics;
    private GraphicsDevice device;
    private SpriteBatch spriteBatch;
    //objekter fra biblioteksklassen
    private QuaternionKamera kamera;
    private ModellBehandling modellering;
    private SpriteBehandling spriteBehandling;
    #endregion

    public StartDemo() {
        graphics = new GraphicsDeviceManager(this);
        Content.RootDirectory = "Content";
        //opprett kobling til kamera klassen
        kamera = new QuaternionKamera(this);
        Components.Add(kamera);
        //opprett kobling til modell klassen
        modellering = new ModellBehandling(this);
        Components.Add(modellering);
        //opprett kobling til spritebehandling klassen
        spriteBehandling = new SpriteBehandling(this);
        Components.Add(spriteBehandling);
    } //konstruktør

    /// <summary>
    /// Allows the game to perform any initialization it needs to before starting
    /// to run. This is where it can query for any required services and load any
    /// non-graphic related content. Calling base.Initialize will enumerate
    /// through any components and initialize them as well.
    /// </summary>
    protected override void Initialize() {
        //oversend verdier til Kamera-klassen
        kamera.ViewAngle = MathHelper.PiOver4;
        kamera.AspectRatio = graphics.GraphicsDevice.Viewport.AspectRatio;
        kamera.NearPlane = 0.5f;
        kamera.FarPlane = 100.0f;
        base.Initialize();
        //Setter størrelse på framebuffer:
        graphics.PreferredBackBufferWidth = 1024;
        graphics.PreferredBackBufferHeight = 768;
        graphics.IsFullScreen = false;
        graphics.ApplyChanges();
        //synliggjør musa
        this.IsMouseVisible = true;
    } //Initialize
}
```

```

/// <summary>
/// LoadContent will be called once per game and is the place to load
/// all of your content.
/// </summary>
protected override void LoadContent() {
    // Create a new SpriteBatch, which can be used to draw textures.
    spriteBatch = new SpriteBatch(GraphicsDevice);
    device = graphics.GraphicsDevice;
} //LoadContent

/// <summary>
/// UnloadContent will be called once per game and is the place to unload
/// all content.
/// </summary>
protected override void UnloadContent() {
    // TODO: Unload any non ContentManager content here
} //UnloadContent

/// <summary>
/// Allows the game to run logic such as updating the world,
/// checking for collisions, gathering input, and playing audio.
/// </summary>
/// <param name="gameTime">Provides a snapshot of timing values.</param>
protected override void Update(GameTime gameTime) {
    base.Update(gameTime);
} //Update

/// <summary>
/// This is called when the game should draw itself.
/// </summary>
/// <param name="gameTime">Provides a snapshot of timing values.</param>
protected override void Draw(GameTime gameTime) {
    //tøm/rens skjermen
    device.Clear(ClearOptions.Target | ClearOptions.DepthBuffer, Color.Black,
                1, 0);
    base.Draw(gameTime);
} //Draw
} //StartDemo
} //namespace

```

FlaggStruktur

```
/// <summary>
/// En struct som inneholder verdier for Flagg. Flagg er en modell
/// som legges ut på veien, som gir bruker poeng ved sammenstøt/kollisjon.
/// Denne struct'n er KUN brukbar til flaggmodellen, da den er lagd med denne
/// hensikten.
/// </summary>
internal struct FlaggStruktur {
    private Model flagg;
    private bool tegnFlagg;
    private Matrix[] flaggMatrise;
    private Vector3 posisjon;

    #region GET & SET METODE
    /// <summary>
    /// Henter/setter modellen
    /// (Model)
    /// </summary>
    internal Model FlaggModell {
        get {
            return flagg;
        }
        set {
            flagg = value;
        }
    }
    /// <summary>
    /// Henter/setter posisjonen hvor flagget skal settes ut
    /// (Vector3)
    /// </summary>
    internal Vector3 Posisjon {
        get {
            return posisjon;
        }
        set {
            posisjon = value;
        }
    }
    /// <summary>
    /// Henter/setter om flagg skal tegnes på skjerm
    /// (bool)
    /// </summary>
    internal bool TegnFlagg {
        get {
            return tegnFlagg;
        }
        set {
            tegnFlagg = value;
        }
    }
}

/// <summary>
/// Henter/setter matrisen som inneholder verdiene til modellen
/// (Matrix[])
/// </summary>
internal Matrix[] FlaggMatrise {
    get {
        return flaggMatrise;
    }
    set {
        flaggMatrise = value;
    }
}
```

```

    }
    #endregion
} //FlaggStruktur

```

LydAvspilling

```

/// <summary>
/// Denne klassen tar seg av alt som har med lyd og musikk å gjøre.
/// </summary>
internal class LydAvspilling : Microsoft.Xna.Framework.DrawableGameComponent {
    private Game game;
    private Song[] bakgrunnsMusikk;
    private SoundEffect bremsing;
    private bool erPauset = false;

    internal LydAvspilling(Game game) : base(game) {
        this.game = game;
    } //konstruktør

    protected override void LoadContent() {
        //opprett array og fyll den med bakgrunnsmusikk
        bakgrunnsMusikk = new Song[3];
        bakgrunnsMusikk[0] = game.Content.Load<Song>(@"Lyd&Musikk\Atmos");
        bakgrunnsMusikk[1] = game.Content.Load<Song>(@"Lyd&Musikk\DrDestru");
        bakgrunnsMusikk[2] = game.Content.Load<Song>(@"Lyd&Musikk\partybie");
        bremsing = game.Content.Load<SoundEffect>(@"Lyd&Musikk\skid");
        //sett at bakgrunnsmusikken skal repeteres (loopes)
        MediaPlayer.IsRepeating = true;
        base.LoadContent();
    } //LoadContent

    /// <summary>
    /// Starter avspilling av bakgrunnsmusikk.
    /// </summary>
    /// <param name="sang">Nummeret (int) på hvilken sang som skal avspilles (1 -
    ///                                     3)</param>
    internal void StartAvspilling(int sang) {
        //trekk fra en, siden array starter på null
        sang--;
        //start avspilling av valgt sang
        MediaPlayer.Play(bakgrunnsMusikk[sang]);
    } //StartAvspilling

    internal void AvspillBremsing() {
        //er spillets lyd slått av?
        if (!Stillhet) {
            //avspill bremselyd
            bremsing.Play();
        } //if (!Stillhet)
    } //AvspillBremsing
}

```

```

public override void Update(GameTime gameTime) {
    //er lyden av og musikken ikke pauset?
    if (Stillhet && !erPauset) {
        //sett musikk på pause
        MediaPlayer.Pause();
        erPauset = true;
    } else if (erPauset && !Stillhet) { //er musikken pauset og lyden på
        //fortsett avspilling av musikk
        MediaPlayer.Resume();
        erPauset = false;
    } //if (!Stillhet)
    base.Update(gameTime);
} //Update

#region GET OG SET METODER
internal static bool Stillhet {
    get {
        return MediaPlayer.IsMuted;
    }
    set {
        MediaPlayer.IsMuted = value;
    }
}
#endregion
} //LydAvspilling

```

PartikkelInformasjon og Partikkeleffekt

```

/// <summary>
/// Struct som inneholder informasjonen/oppbyggingen av partikkelen(e)
/// </summary>
internal struct PartikkelInformasjon {
    public float PartikkelSkapt;
    public float MaxAlder;
    public Vector2 StartPosisjon;
    public Vector2 Akselerasjon;
    public Vector2 Retning;
    public Vector2 Posisjon;
    public float Skalering;
    public Color Farge;
} //PartikkelInformasjon

/// <summary>
/// Denne klassen tar for seg partikkeleffekter.
/// Grunnlaget er basert på oppsett fra Riemer;
/// http://www.riemers.net/eng/Tutorials/XNA/Csharp/Series2D/Particles.php
/// </summary>
public class Partikkeleffekt : Microsoft.Xna.Framework.DrawableGameComponent {
    private SpriteBatch spriteBatch;
    private Game game;
    //teksturen som inneholder bilde av en eksplosjon
    private Texture2D partikkelTekstur;
    //liste som skal inneholde partiklene
    private List<PartikkelInformasjon> partikkelListe = new
List<PartikkelInformasjon>();
    private Random randomizer = new Random();

```

```

/// <summary>
/// Konstruktøren til PartikkelEffekt klassen
/// </summary>
/// <param name="game">Objekt av Game</param>
public Partikkeleffekt(Game game)
    : base(game) {
    this.game = game;
} //konstruktør

/// <summary>
/// Oppretter/skaper en partikkeleffekt
/// </summary>
/// <param name="startPosisjon">Startposisjonen for partikkeleffekten</param>
/// <param name="antallPartikler">Antall partikler som skal lages</param>
/// <param name="str1">Størrelsen på partikkelen</param>
/// <param name="maxAlder">Partikkelens levetid</param>
/// <param name="gameTime">Objekt av gameTime</param>
internal void SkapPartikkeleffekt(Vector2 startPosisjon, int antallPartikler,
    float str1, float maxAlder, gameTime gameTime) {
    //løkke som skaper/oppretter partikler
    for (int i = 0; i < antallPartikler; i++) {
        OpprettPartikkel(startPosisjon, str1, maxAlder, gameTime);
    } //for
} //SkapPartikkeleffekt

private void OpprettPartikkel(Vector2 startPosisjon, float eksplosjonStr1,
    float maxAlder, gameTime gameTime) {
    //opprett et partikkel objekt
    PartikkelInformasjon partikkel = new PartikkelInformasjon();
    //sett startpoisjon for partikkelen
    partikkel.StartPosisjon = startPosisjon;
    partikkel.Posisjon = partikkel.StartPosisjon;
    //sett når partikkelen ble skapt
    partikkel.PartikkelSkapt =
(float)gameTime.TotalGameTime.TotalMilliseconds;
    partikkel.MaxAlder = maxAlder;
    partikkel.Skalering = 0.25f;
    partikkel.Farge = Color.White;
    //randomiser partikkelens distanse
    float partikkelDistanse = (float)randomizer.NextDouble() * eksplosjonStr1;
    Vector2 displacement = new Vector2(partikkelDistanse, 0);
    float vinkel = MathHelper.ToRadians(randomizer.Next(360));
    displacement = Vector2.Transform(displacement,
        Matrix.CreateRotationZ(vinkel));

    //sett retning og akselerasjon
    partikkel.Retning = displacement * 2.0f;
    partikkel.Akselerasjon = -partikkel.Retning;
    //legg til partikkel i listen
    partikkelListe.Add(partikkel);
} //OpprettPartikkel

/// <summary>
/// LoadContent will be called once per game and is the place to load
/// all of your content.
/// </summary>
protected override void LoadContent() {
    // Create a new SpriteBatch, which can be used to draw textures.
    spriteBatch = new SpriteBatch(GraphicsDevice);
    partikkelTekstur = game.Content.Load<Texture2D>(@"Sprites\explosion");
} //LoadContent

/// <summary>

```

```

/// UnloadContent will be called once per game and is the place to unload
/// all content.
/// </summary>
protected override void UnloadContent() {
    // TODO: Unload any non ContentManager content here
} //UnloadContent

/// <summary>
/// Allows the game to run logic such as updating the world,
/// checking for collisions, gathering input, and playing audio.
/// </summary>
/// <param name="gameTime">Provides a snapshot of timing values.</param>
public override void Update(GameTime gameTime) {
    //finnes det partikler i listen?
    if (partikkelliste.Count > 0) {
        //oppdater partiklene som finnes i listen
        OppdaterPartikler(gameTime);
    } //if (partikkelliste.Count > 0)
    base.Update(gameTime);
} //Update

private void OppdaterPartikler(GameTime gameTime) {
    //hent ut gjeldende tidspunkt
    float naaTid = (float)gameTime.TotalGameTime.TotalMilliseconds;
    //loop gjennom eksisterende partikler i listen
    for (int i = partikkelliste.Count - 1; i >= 0; i--) {
        //hent partikkel fra listen
        PartikkelInformasjon partikkel = partikkelliste[i];
        float levetid = naaTid - partikkel.PartikkelSkapt;
        //har partikkelen levd over sin gitte levealder?
        if (levetid > partikkel.MaxAlder) {
            //fjern partikkel fra listen
            partikkelliste.RemoveAt(i);
        } else {
            float relAge = levetid / partikkel.MaxAlder;
            //flytt partikkel
            partikkel.Posisjon = 0.5f * partikkel.Akselerasjon * relAge *
                relAge + partikkel.Retning * relAge +
                partikkel.StartPosisjon;
            //sett ny farge på partikkel basert på alder
            float invAge = 1.0f - relAge;
            partikkel.Farge = new Color(new Vector4(invAge, invAge, invAge,
                invAge));
            //hent ut hvor mye partikkelen har forflyttet seg
            Vector2 posisjonFraSenter = partikkel.Posisjon -
                partikkel.StartPosisjon;
            float distance = posisjonFraSenter.Length();
            partikkel.Skalering = (50.0f + distance) / 200.0f;
            //legg oppdatert partikkel tilbake i listen
            partikkelliste[i] = partikkel;
        } //if (levetid > partikkel.MaxAlder)
    } //for
} //OppdaterPartikler

```

```

/// <summary>
/// This is called when the game should draw itself.
/// </summary>
/// <param name="gameTime">Provides a snapshot of timing values.</param>
public override void Draw(GameTime gameTime) {
    //tegn tekstur uten den svarte bakgrunnen
    spriteBatch.Begin(SpriteSortMode.Deferred, BlendState.Additive);
    TegnEksplasjon();
    spriteBatch.End();
    //tilbakestill verdier etter å ha brukt SpriteBatch
    GraphicsDevice.BlendState = BlendState.Opaque;
    GraphicsDevice.DepthStencilState = DepthStencilState.Default;
    GraphicsDevice.SamplerStates[0] = SamplerState.LinearWrap;
    base.Draw(gameTime);
} //Draw

private void TegnEksplasjon() {
    for (int i = 0; i < partikkelListe.Count; i++) {
        PartikkelInformasjon partikkel = partikkelListe[i];
        //tegn partikkelen
        spriteBatch.Draw(partikkelTekstur, partikkel.Posisjon, null,
            partikkel.Farge, i, new Vector2(256, 256),
            partikkel.Skalering, SpriteEffects.None, 1);
    } //for
} //TegnEksplasjon
} //PartikkelEffekter

```


SpriteBehandling

```
/// <summary>
/// Denne klassen tar for seg alt som har med Sprites å gjøre.
/// Dette gjelder ting som tekst, bilder o.l.
/// </summary>
public class SpriteBehandling : Microsoft.Xna.Framework.DrawableGameComponent {
    private Game game;
    private SpriteBatch spriteBatch;
    private Texture2D girSkifte;
    private SpriteFont arialFont, boldArialFont;
    private Partikkeleffekt partikkelEffekt;
    private static int aktivSang;

    /// <summary>
    /// Konstruktøren til SpriteBehandling
    /// </summary>
    /// <param name="game">Objekt av Game</param>
    public SpriteBehandling(Game game)
        : base(game) {
        this.game = game;
        //opprett kobling til partikkeleffekt klassen
        partikkelEffekt = new Partikkeleffekt(game);
        game.Components.Add(partikkelEffekt);
    } //konstruktør

    /// <summary>
    /// Allows the game component to perform any initialization it needs to before
    /// starting to run. This is where it can query for any required services and
    /// load content.</summary>
    public override void Initialize() {
        base.Initialize();
    } //Initialize

    /// <summary>
    /// Laster inn Sprites
    /// </summary>
    protected override void LoadContent() {
        spriteBatch = new SpriteBatch(GraphicsDevice);
        arialFont = game.Content.Load<SpriteFont>(@"Sprites\ArialFont");
        boldArialFont = game.Content.Load<SpriteFont>(@"Sprites\BoldArialFont");
        girSkifte = game.Content.Load<Texture2D>(@"Sprites\Girskifte2");
        base.LoadContent();
    } //LoadContent

    /// <summary>
    /// Allows the game component to update itself.
    /// </summary>
    /// <param name="gameTime">Provides a snapshot of timing values.</param>
    public override void Update(GameTime gameTime) {
        //vises hele bilen?
        if (QuaternionKamera.Synsvinkel == 1) {
            //skap en partikkeleffekt som tegnes bak eksospotta (gir inntrykk av
            //eksos)
            partikkelEffekt.SkapPartikkeleffekt(new Vector2(559.0f, 654.0f), 1,
                4.0f, 100.0f, gameTime);
        } //if (kamera.Synsvinkel == 1)
        base.Update(gameTime);
    } //Update
```

```

/// <summary>
/// Utfører selve tegningen av modellene.

/// Overrides Draw(GameTime)
/// </summary>
/// <param name="gameTime"></param>
public override void Draw(GameTime gameTime) {
    TegnGirskifte();
    VisInformasjonPaaSkjerm();
    base.Draw(gameTime);
} //Draw

#region TEKST PÅ SKJERM
/// <summary>
/// Oppretter forskjellige tekststrenger som skal vises på skjerm
/// </summary>
public void VisInformasjonPaaSkjerm() {
    string valgtGir, valgtSynsvinkel;
    //hvilken synsvinkel er aktiv?
    if (QuaternionKamera.Synsvinkel == 1) {
        valgtSynsvinkel = "Vis bil";
    } else {
        valgtSynsvinkel = "Frontvindu";
    } // if (kamera.Synsvinkel == 1)
    //er aktivt gir Revers?
    if (QuaternionKamera.GirNivaa == QuaternionKamera.REVERS) {
        //sett string valgtGir med teksten Revers
        valgtGir = "Revers";
    } else { //ikke revers, vis girets tallverdi
        valgtGir = QuaternionKamera.GirNivaa.ToString();
    } //if (kamera.VisGirNivaa == QuaternionKamera.REVERS)
    //informasjonstekst som vises øverst i venstre hjørne
    string info = "Poeng: " + ModellBehandling.Poeng.ToString() + "\n"
        + "Synsvinkel: " + valgtSynsvinkel + "\n"
        + "Bakgrunnsmusikk valgt: " + AktivMusikk + "\n";
    //er lyden av?
    if (LydAvspilling.Stillhet) {
        info += "Lyd: Lyd er av";
    } else {
        info += "Lyd: Lyd er paa";
    } //if (LydAvspilling.Stillhet)
    //skriv ut tekst på skjerm
    VisTekstPaaSkjerm(arialFont, info, 1, 1, Color.Black, Color.White);
    //tekst som vises nederst til venstre (rødt for bedre synlighet mot veien)
    VisTekstPaaSkjerm(arialFont, "Trykk H eller F1 for hjelp", 1, 740,
        Color.White, Color.Red);
}

```

```

//skal hjelp vises?
if (QuaternionKamera.VisHjelp) {
    string hjelpetekst = "Hjelpemeny (trykk H eller F1 for aa skjule
                          Hjelp):";
    VisTekstPaaSkjerm(boldArialFont, hjelpetekst, 253, 50, Color.Black,
                     Color.Gold);
    hjelpetekst = "- For aa bevege bilen: Bruk piltastene eller W-A-S-
                  D\n"
                  + "- For aa sette bilen i revers: Trykk S eller Pil
                    Ned\n"
                  + "- For aa bremse: Trykk Space (Mellomrom)\n"
                  + "- For aa gire opp: Trykk Q\n"
                  + "- For aa gire ned: Trykk E\n"
                  + "- For aa skifte synsvinkel: Trykk Z\n"
                  + "- For aa starte nytt spill: Trykk R\n"
                  + "- For aa avslutte: Trykk ESC\n"
                  + "- For aa veksle mellom bakgrunnsmusikk: Trykk 1, 2
                    eller 3\n"
                  + "- For aa slaa lyd av: Trykk M\n"
                  + "\n"
                  + " - Du faar poeng for hvert flagg du kjorer over\n"
                  + "   (men poengsum nullstilles ved nytt spill).";
    VisTekstPaaSkjerm(arialFont, hjelpetekst, 253,80, Color.Black,
                     Color.Gold);
    //tekst som vises over giret (rødt for bedre synlighet mot veien)
    hjelpetekst = "Dette er aktivt gir:";
    VisTekstPaaSkjerm(arialFont, hjelpetekst, 796, 629, Color.White,
                     Color.Red);
} //if (kamera.VisHjelp)
} //visInformasjonPaaSkjerm

private void VisTekstPaaSkjerm(SpriteFont font, string tekst, int x, int y,
                             Color bakgrunnsFarge, Color tekstFarge) {
    //Skriver tekst:
    spriteBatch.Begin();
    //Skriver teksten to ganger, først med svart bakgrunn og deretter
    //med hvitt, en piksel ned og til venstre, slik at teksten blir mer
    //lesbar.
    spriteBatch.DrawString(font, tekst, new Vector2(x, y), bakgrunnsFarge);
    spriteBatch.DrawString(font, tekst, new Vector2(x - 1, y - 1),
                           tekstFarge);
    spriteBatch.End();
    //Tilbakestilling av parametere etter bruk, jfr. Shawn Hargreaves:
    //http://blogs.msdn.com/b/shawnhar/archive/2010/06/18/spritebatch-and-
    //renderstates-in-xna-game-studio-4-0.aspx
    GraphicsDevice.BlendState = BlendState.Opaque;
    GraphicsDevice.DepthStencilState = DepthStencilState.Default;
    GraphicsDevice.SamplerStates[0] = SamplerState.LinearWrap;
} //visTekstPaaSkjerm
#endregion

```

```

#region BILDE PÅ SKJERM
//basert på kodeeksempel fra XNA Game Studio 4.0 Programming s.22
private void TegnGirskifte() {
    //sortering av billedeler skjer fra øverst til venstre til øverst til
    // høyre, deretter midten av venstre til midten av høyre
    spriteBatch.Begin(SpriteSortMode.FrontToBack, null);
    //henter ut girnivå
    int gir = QuaternionKamera.GirNivaa - 1;
    //opprett et rektangel som inneholder "rammen" til bildet
    Rectangle rektangel = new Rectangle((gir % 3) * (girSkifte.Width / 3),
        //beregner y basert på hvilken del av bildet som blir tegnet
        // (øvre/nedre del)
        (gir < 3) ? 0 : (girSkifte.Height / 2),
        //rektangelets bredde og høyde (deler på 3 siden det er tre bilder
        // sidelengs)
        girSkifte.Width / 3, girSkifte.Height / 2);
    //tegner bildet på skjerm, nederst i høyre hjørne
    spriteBatch.Draw(girSkifte, new Vector2(800, 660), rektangel, Color.White,
        0.0f, Vector2.Zero, 1.0f, SpriteEffects.None, 0);
    spriteBatch.End();
} //TegnGirskifte
#endregion

/// <summary>
/// Henter/setter en tallverdi for hvilken sang som er aktiv
/// </summary>
internal static int AktivMusikk {
    get {
        return aktivSang;
    }
    set {
        aktivSang = value;
    }
}
} //SpriteBehandling

```

Modellbehandling

```
/// <summary>
/// Denne klassen tar for seg tegning av modeller på skjerm og i tillegg har den
/// kollisjonsdeteksjon. Kollisjonsdeteksjonen sjekker om bil og flagg kolliderer,
/// og hvis de kolliderer så genereres poeng og flagg fjernes fra skjerm.
/// </summary>
public class ModellBehandling : Microsoft.Xna.Framework.DrawableGameComponent {
    #region VARIABLE
    private Game game;
    //innlasting av Skybox og modeller
    private Model skyboxModell;
    private Model bilModell;
    //tilhørende matriser
    private Matrix[] flaggMatrise;
    private Matrix[] bilMatrise;
    //array som inneholder informasjon om flagg
    private FlaggStruktur[] flaggArray;
    //array som inneholder koordinater for hvor flagg skal plasseres ut
    private Vector3[] flaggKoordinater;
    //verdier for kollisjon og poeng
    private static int poeng;
    private int resterendeFlagg = 10;
    #endregion

    /// <summary>
    /// Konstruktøren til ModellBehandling
    /// </summary>
    /// <param name="game">Objekt av Game</param>
    public ModellBehandling(Game game)
        : base(game) {
        this.game = game;
    } //konstruktør

    /// <summary>
    /// Allows the game component to perform any initialization it needs to before
    /// starting to run. This is where it can query for any required services and
    /// load content.</summary>
    public override void Initialize() {
        float y = 0.0f;
        flaggKoordinater = new Vector3[10];
        flaggKoordinater[0] = new Vector3(-14.0f, y, -20.0f);
        flaggKoordinater[1] = new Vector3(-23.0f, y, -12.0f);
        flaggKoordinater[2] = new Vector3(-18.0f, y, 10.0f);
        flaggKoordinater[3] = new Vector3(21.0f, y, 14.0f);
        flaggKoordinater[4] = new Vector3(24.0f, y, -9.0f);
        flaggKoordinater[5] = new Vector3(0.0f, y, -27.0f);
        flaggKoordinater[6] = new Vector3(33.0f, y, 12.0f);
        flaggKoordinater[7] = new Vector3(10.0f, y, 29.0f);
        flaggKoordinater[8] = new Vector3(-30.0f, y, 31.0f);
        flaggKoordinater[9] = new Vector3(15.0f, y, 19.0f);
        base.Initialize();
    } //Initialize
}
```

```

/// <summary>
/// Override LoadContent;
/// Laster inn modellene som skal brukes
/// </summary>
protected override void LoadContent() {
    //last inn modeller
    skyboxModell = game.Content.Load<Model>(@"Modeller\Tribune");
    bilModell = LoadModelWithBoundingSphere(@"Modeller\Bil", ref bilMatrise);
    //opprett array
    flaggArray = new FlaggStruktur[10];
    //løkke som fyller array'n med verdier
    for (int i = 0; i < 10; i++) {
        //opprett et objekt av strukturen
        FlaggStruktur flagg = new FlaggStruktur();
        //sett verdiene til strukturen
        flagg.FlaggModell = LoadModelWithBoundingSphere(@"Modeller\Flagg", ref
                                                                flaggMatrise);

        flagg.FlaggMatrise = flaggMatrise;
        //henter ut gjeldende flaggs posisjon basert på satt koordinat i array
        flagg.Posisjon = flaggKoordinater[i];
        flagg.TegnFlagg = true;
        //legg objektet av strukturen i array
        flaggArray[i] = flagg;
    } //for
    base.LoadContent();
} //LoadContent

//tatt fra forelesningsnotat "Del14 - Kollisjonstest"
private Model LoadModelWithBoundingSphere(String modelName, ref Matrix[]
                                          matrix) {
    Model model = game.Content.Load<Model>(modelName);
    matrix = new Matrix[model.Bones.Count];
    //Legger komplett transformasjonsmatrise for hver ModelMesh i
    matrisetabellen:
    model.CopyAbsoluteBoneTransformsTo(matrix);
    //Finner BoundingSphere for hele modellen:
    BoundingSphere completeBoundingSphere = new BoundingSphere();
    foreach (ModelMesh mesh in model.Meshes) {
        //Henter ut BoudigSphere for aktuell ModelMesh:
        BoundingSphere origMeshSphere = mesh.BoundingSphere;
        //Denne transformerer i forhold til sitt Bone:
        origMeshSphere = XNAUtils.TransformBoundingSphere(origMeshSphere,
        matrix[mesh.ParentBone.Index]);
        //Slår sammen:
        completeBoundingSphere =
    BoundingSphere.CreateMerged(completeBoundingSphere, origMeshSphere);
    } //foreach
    model.Tag = completeBoundingSphere;
    return model;
} //LoadModelWithBoundingSphere

```

```

/// <summary>
/// Allows the game component to update itself.
/// </summary>
/// <param name="gameTime">Provides a snapshot of timing values.</param>
public override void Update(GameTime gameTime) {
    //har bruker startet nytt spill?
    if (QuaternionKamera.StartNyttSpill) {
        //nullstill poengsum
        Poeng = 0;
        //sett at alle flagg er på skjerm
        resterendeFlagg = 10;
        //loop gjennom array og sett at alle flaggene skal tegnes
        for (int i = 0; i < flaggArray.Length; i++) {
            flaggArray[i].TegnFlagg = true;
        } //for
        //sett at nytt spill har startet
        QuaternionKamera.StartNyttSpill = false;
    } //if (kamera.StartNyttSpill)
    base.Update(gameTime);
} //Update

/// <summary>
/// Utfører selve tegningen av modellene.
/// Overrides Draw(GameTime)
/// </summary>
/// <param name="gameTime">Objekt av GameTime</param>
public override void Draw(GameTime gameTime) {
    OpprettSkybox();
    //siden det kun er to verdier for zoom, så settes det her at
    //bilen kun skal tegnes dersom bruker ønsker å se bilen
    //(dersom det blir lagt til flere, må denne oppdateres/endres)
    if (QuaternionKamera.Synsvinkel == 1) {
        //legg ut bilmodellen på skjermen
        TegnBil();
    } //if (kamera.Synsvinkel == 1)
    //sjekk om bil og flagg har kollidert
    SjekkEtterKollisjon();
    //tegn flagg på skjerm
    TegnFlagg();
    base.Draw(gameTime);
} //Draw

#region KOLLISJONSMETODER
private void SjekkEtterKollisjon() {
    //loop gjennom alle flaggene som finnes i array
    for (int i = 0; i < flaggArray.Length; i++) {
        FlaggStruktur flagg = flaggArray[i];
        //kjør if-test kun på flagg som skal tegnes på skjerm
        if (flagg.TegnFlagg) {
            //world matrisen for flagg, skrevet på egen linje for bedre
            oversikt
            Matrix worldMatrixFlagg = ReturnWorldMatrixFlagg(flagg.Posisjon);
            //hent verdi for kollisjon
            bool modellerKollidert = FinkornetModellKollisjon(bilModell,
                ReturnWorldMatrixBil,
                flagg.FlaggModell, worldMatrixFlagg);
            //sjekk om bil og flagg kolliderer
            if (modellerKollidert) {
                //øk poeng
                Poeng++;
                //trekk fra resterende flagg
                resterendeFlagg--;
            }
        }
    }
}

```

```

        //sett at dette flagget ikke skal tegnes, m.a.o. det er
        påkjørt
        flaggArray[i].TegnFlagg = false;
    } //if (modellerKollidert)
} //if (flagg.TegnFlagg)
} //for
//er alle flaggene sanket inn?
if (resterendeFlagg == 0) {
    //loop gjennom array og sett at alle flaggene skal tegnes
    for (int i = 0; i < flaggArray.Length; i++) {
        flaggArray[i].TegnFlagg = true;
    } //for
    resterendeFlagg = 10;
} //if (resterendeFlagg == 0)
} //SjekkEtterKollisjon

//tatt fra XNA Game Programming Recipes s. 321
private bool ModellKollisjon(Model modell1, Matrix world1, Model model2, Matrix
    world2) {
    //henter den originale boundsphere
    BoundingBox origSphere1 = (BoundingBox)modell1.Tag;
    //oppretter en ny boundsphere basert på modellens forflytning
    BoundingBox sphere1 = XNAUtils.TransformBoundingBox(origSphere1,
        world1);
    BoundingBox origSphere2 = (BoundingBox)model2.Tag;
    BoundingBox sphere2 = XNAUtils.TransformBoundingBox(origSphere2,
        world2);
    //henter verdi basert på om modellene etter forflytning kolliderer
    bool collision = sphere1.Intersects(sphere2);
    return collision;
} //ModelsCollide

```



```

//basert på XNA Game Programming Recipes s.325
private bool FinkornetModellKollisjon(Model modell1, Matrix world1, Model
modell2, Matrix world2) {
    //sjekk om de globale boundsphere kolliderer
    if (ModellKollisjon(modell1, world1, modell2, world2) == false) {
        return false;
    } //if (ModelsCollide(...))
    //opprett variabler med verdiene til modell1
    Matrix[] modell1Transforms = new Matrix[modell1.Bones.Count];
    modell1.CopyAbsoluteBoneTransformsTo(modell1Transforms);
    //array som inneholder
    BoundingSphere[] modell1Spheres = new BoundingSphere[modell1.Meshes.Count];
    //loop gjennom og opprett boundsphere for hver mesh
    for (int i = 0; i < modell1.Meshes.Count; i++) {
        ModelMesh mesh = modell1.Meshes[i];
        BoundingSphere origSphere = mesh.BoundingSphere;
        Matrix trans = modell1Transforms[mesh.ParentBone.Index] * world1;
        BoundingSphere transSphere =
XNAUtils.TransformBoundingSphere(origSphere, trans);
        //legg hver mesh sin boundsphere inn i array
        modell1Spheres[i] = transSphere;
    } //for
    //gjør det samme som over for modell2
    Matrix[] modell2Transforms = new Matrix[modell2.Bones.Count];
    modell2.CopyAbsoluteBoneTransformsTo(modell2Transforms);
    BoundingSphere[] modell2Spheres = new BoundingSphere[modell2.Meshes.Count];
    //loop gjennom og opprett boundsphere for hver mesh
    for (int i = 0; i < modell2.Meshes.Count; i++) {
        ModelMesh mesh = modell2.Meshes[i];
        BoundingSphere origSphere = mesh.BoundingSphere;
        Matrix trans = modell2Transforms[mesh.ParentBone.Index] * world2;
        BoundingSphere transSphere =
XNAUtils.TransformBoundingSphere(origSphere, trans);
        modell2Spheres[i] = transSphere;
    } //for
    //variabel for om kollisjon har oppstått
    bool collision = false;
    //loop gjennom array'ene med boundspheres
    for (int i = 0; i < modell1Spheres.Length; i++) {
        for (int j = 0; j < modell2Spheres.Length; j++) {
            //har modellene kollidert?
            if (modell1Spheres[i].Intersects(modell2Spheres[j])) {
                collision = true;
            } //if (transSphere1.Intersects(transSphere2))
        } //indre for
    } //ytre for
    return collision;
} //ModelsCollideFineCheck
#endregion

```

```

#region TEGN MODELLER
private void OpprettSkybox() {
    Matrix[] skyMatrise = new Matrix[skyboxModell.Bones.Count];
    skyboxModell.CopyAbsoluteBoneTransformsTo(skyMatrise);
    //start å tegne modellen (skybox)
    foreach (ModelMesh mesh in skyboxModell.Meshes) {
        foreach (BasicEffect effect in mesh.Effects) {
            //setter opp skyboxen
            effect.World = skyMatrise[mesh.ParentBone.Index] *
                ReturnWorldMatrix;
            effect.View = QuaternionKamera.ViewMatrix;
            effect.Projection = QuaternionKamera.ProjectionMatrix;
            //slår på lys
            effect.LightingEnabled = true;
            effect.AmbientLightColor = new Vector3(0.1f, 0.1f, 0.1f);
            effect.PreferPerPixelLighting = true;
            //setter på lysretning og retning for lyset
            effect.DirectionalLight0.Direction = new Vector3(-1, -1, -1);
            effect.DirectionalLight0.DiffuseColor = Color.White.ToVector3();
            effect.DirectionalLight0.Enabled = true;
            effect.DirectionalLight1.Enabled = true;
            effect.DirectionalLight2.Enabled = false;
        } //indre foreach
        mesh.Draw();
    } //ytte foreach
} //opprettSkybox

private void TegnBil() {
    //start å tegne modellen (bilen)
    foreach (ModelMesh mesh in bilModell.Meshes) {
        foreach (BasicEffect effect in mesh.Effects) {
            effect.EnableDefaultLighting();
            effect.World = bilMatrise[mesh.ParentBone.Index] *
                ReturnWorldMatrixBil;
            effect.View = QuaternionKamera.ViewMatrix;
            effect.Projection = QuaternionKamera.ProjectionMatrix;
        } //indre foreach
        mesh.Draw();
    } //ytte foreach
} //tegnBil

private void TegnFlagg() {
    //loop gjennom alle flagg som ligger i array
    foreach (FlaggStruktur flagg in flaggArray) {
        //skal flagget tegnes på skjerm?
        if (flagg.TegnFlagg) {
            //start å tegne modellen (flagg)
            foreach (ModelMesh mesh in flagg.FlaggModell.Meshes) {
                foreach (BasicEffect effect in mesh.Effects) {
                    effect.EnableDefaultLighting();
                    effect.World = flagg.FlaggMatrise[mesh.ParentBone.Index] *
                        ReturnWorldMatrixFlagg(flagg.Posisjon);
                    effect.View = QuaternionKamera.ViewMatrix;
                    effect.Projection = QuaternionKamera.ProjectionMatrix;
                } //indre foreach
                mesh.Draw();
            } //ytte foreach
        } //
    } //foreach
} //tegnFlagg
#endregion

```

```

#region GET METODER
private Matrix ReturnWorldMatrixBil {
    get {
        Matrix skalering = Matrix.CreateScale(1.0f / 350.0f);
        Matrix rotasjon =
Matrix.CreateFromQuaternion(QuaternionKamera.Rotasjon);
        Matrix translasjon =
Matrix.CreateTranslation(QuaternionKamera.Posisjon);
        //opprett worldmatrise basert på ovenstående matriser
        Matrix worldMatrix = skalering * rotasjon * translasjon;
        return worldMatrix;
    }
}

private Matrix ReturnWorldMatrix {
    get {
        Matrix worldMatrix = Matrix.CreateScale(0.01f, 0.01f, 0.01f);
        return worldMatrix;
    }
}

private Matrix ReturnWorldMatrixFlagg(Vector3 posisjon) {
    //beregner translasjon før skalering
    //(flagg skal settes ut på forskjellige plasser)
    Matrix worldMatrix = ReturnWorldMatrix *
Matrix.CreateTranslation(posisjon);
    return worldMatrix;
}

/// <summary>
/// Returnerer brukers poengsum
/// </summary>
internal static int Poeng {
    get {
        return poeng;
    }
    private set {
        poeng = value;
    }
}
}
#endregion
} //ModellBehandling

```

QuaternionKamera

```
/// <summary>
/// Dette er en Kamera-klasse basert på Quaternion, som i tillegg
/// håndterer input fra bruker (tastaturinput).
///<para>
/// Dette betyr at klassen har ansvar for posisjonering og forflytting
/// både av kamera og bilen som bruker kontrollerer.
/// </para>
/// For å unngå problemer med arv og overstyring av metoder er klassen forseglet
/// (sealed).Klassen er delvis basert på Riemers Quaternion s. 50 - 57 i Game XNA
/// 3.0 Programming Recipes </summary>
public sealed class QuaternionKamera : Microsoft.Xna.Framework.GameComponent {
    #region VARIABLER
    //konstanter
    private const int MIN_GIR = 1;
    private const int MAX_GIR = 4;
    /// <summary>
    /// Verdien for Gir: Revers
    /// </summary>
    public const int REVERS = 5;
    private const int MAX_ZOOM = 2;
    private GraphicsDevice device;
    //vektorer
    private static Vector3 posisjon;
    //matriser
    private static Matrix projectionMatrix;
    private static Matrix viewMatrix;
    private static Quaternion kameraRotasjon;
    //verdier for kamera og forflytning
    private float viewAngle;
    private float aspectRatio;
    private float nearPlane;
    private float farPlane;
    private static int girNivaa;
    private int forflytning = 0;
    private static int zoom = 1;
    //verdi for om bruker ønsker å starte et nytt spill
    private static bool nyttSpill = true;
    //KeyboardState som tar vare på hvilken knapp som ble sist trykket
    //(dette for å unngå f.eks. at giring går fra 1 -> 4 på et tastetrykk)
    //funnet via VS Hjelp: F1
    private KeyboardState forrigeTast;
    private static bool visHjelp;
    //lydkontroll variabler
    private bool muteLyd;
    private LydAvspilling lydAvpilling;
    #endregion

    /// <summary>
    /// Konstruktøren til QuaternionKamera, her opprettes en kobling
    /// til klassen LydAvspilling som håndterer lyd og musikk.
    /// </summary>
    /// <param name="game">Objekt av Game</param>
    public QuaternionKamera(Game game)
        : base(game) {
        lydAvpilling = new LydAvspilling(game);
        game.Components.Add(lydAvpilling);
    } //konstruktør
```

```

/// <summary>
/// Allows the game component to perform any initialization it needs to before
/// starting to run. This is where it can query for any required services and
/// load content.</summary>
public override void Initialize() {
    device = Game.GraphicsDevice;
    base.Initialize();
    //opprett projectionmatrix basert på Properties
    projectionMatrix = Matrix.CreatePerspectiveFieldOfView(ViewAngle,
AspectRatio, NearPlane, FarPlane);
    ResetSpill();
} //Initialize

private void ResetSpill() {
    //stopper bilen
    forflytning = 0;
    //setter forrige tast til gjeldende
    forrigeTast = Keyboard.GetState();
    //setter gir til første gir
    GirNivaa = MIN_GIR;
    Synsvinkel = 1;
    VisHjelp = false;
    StartNyttSpill = true;
    //setter startposisjon for kamera i "spillverdenen"
    posisjon = new Vector3(-25.0f, 0.40f, -6.5f);
    //resetter/tømmer kameraRotasjon
    kameraRotasjon = Quaternion.Identity;
    //start avspilling av sang1
    lydAvpilling.StartAvspilling(1);
    //oversend til SpriteBehandling at sang1 avspilles
    SpriteBehandling.AktivMusikk = 1;
    //oppdaterer view matrisen
    OppdaterViewMatrix();
} //resetSpill

/// <summary>
/// Allows the game component to update itself.
/// </summary>
/// <param name="gameTime">Provides a snapshot of timing values.</param>
public override void Update(GameTime gameTime) {
    //les tastatur input fra bruker
    UpdateInput();
    //sjekk om kamera/bil krasjer med skybox
    KrasjerMedSkybox();
    base.Update(gameTime);
} //Update

```

```

#region UPDATES
/// <summary>
/// Oppdaterer kameraposisjon og forflytning basert på hvilken tast som ble
/// trykket, og det sjekkes også at gjeldende tast ikke er holdt inne
/// (dette for å unngå at gir går fra 1 -> 4 på et tastetrykk)
/// </summary>
private void UpdateInput() {
    float venstrehoyreRotasjon = 0.0f;
    //sjekk hvilken knapp som ble trykket
    KeyboardState nyTast = Keyboard.GetState();
    //ønsker bruker å avslutte?
    if (nyTast.IsKeyDown(Keys.Escape)) {
        //avslutter og lukker programmet
        Game.Exit();
    } //if (nyTast.IsKeyDown(Keys.Escape))

    #region AVSPILLING AV LYD/MUSIKK
    //er tast 1 trykket; isåfall start avspilling av sang1
    if ((nyTast.IsKeyDown(Keys.D1) && !forrigeTast.IsKeyDown(Keys.D1))
        || (nyTast.IsKeyDown(Keys.NumPad1) &&
!forrigeTast.IsKeyDown(Keys.NumPad1))) {
        //start avspilling av sang1
        lydAvpilling.StartAvspilling(1);
        //oversend til SpriteBehandling at sang1 avspilles
        SpriteBehandling.AktivMusikk = 1;
    } //if (nyTast.IsKeyDown(Keys.D1) || nyTast.IsKeyDown(Keys.NumPad1))

    if ((nyTast.IsKeyDown(Keys.D2) && !forrigeTast.IsKeyDown(Keys.D2))
        || (nyTast.IsKeyDown(Keys.NumPad2) &&
!forrigeTast.IsKeyDown(Keys.NumPad2))) {
        //start avspilling av sang1
        lydAvpilling.StartAvspilling(2);
        //oversend til SpriteBehandling at sang2 avspilles
        SpriteBehandling.AktivMusikk = 2;
    } //if (nyTast.IsKeyDown(Keys.D2) || nyTast.IsKeyDown(Keys.NumPad2))

    if ((nyTast.IsKeyDown(Keys.D3) && !forrigeTast.IsKeyDown(Keys.D3))
        || (nyTast.IsKeyDown(Keys.NumPad3) &&
!forrigeTast.IsKeyDown(Keys.NumPad3))) {
        //start avspilling av sang1
        lydAvpilling.StartAvspilling(3);
        //oversend til SpriteBehandling at sang3 avspilles
        SpriteBehandling.AktivMusikk = 3;
    } //if (nyTast.IsKeyDown(Keys.D3) || nyTast.IsKeyDown(Keys.NumPad3))

    //skal lyd slås av/på?
    if (nyTast.IsKeyDown(Keys.M) && !forrigeTast.IsKeyDown(Keys.M)) {
        muteLyd = !muteLyd;
        lydAvspilling.Stillhet = muteLyd;
    } //if (nyTast.IsKeyDown(Keys.M) && !forrigeTast.IsKeyDown(Keys.M))
    #endregion

    //skal bilen stoppe (bremse)?
    if (nyTast.IsKeyDown(Keys.Space) && !forrigeTast.IsKeyDown(Keys.Space)) {
        //stopp forflytning
        forflytning = 0;
        //sett bilen i første gir
        GirNivaa = MIN_GIR;
        //spill av bremselyd
        lydAvpilling.AvspillBremsing();
    } //if (nyTast.IsKeyDown(Keys.Space))

```

```

#region FLYTT BIL FREMVER
//skal bilen flyttes fremover?
if (nyTast.IsKeyDown(Keys.Up) && !forrigeTast.IsKeyDown(Keys.Up)) {
    forflytning = -1;
    //er nåværende gir revers?
    if (GirNivaa == REVERS) {
        //sett bilen i første gir
        GirNivaa = MIN_GIR;
    } //if (GirNivaa == REVERS)
} //if (nyTast.IsKeyDown(Keys.Up))
if (nyTast.IsKeyDown(Keys.W) && !forrigeTast.IsKeyDown(Keys.W)) {
    forflytning = -1;
    //er nåværende gir revers?
    if (GirNivaa == REVERS) {
        //sett bilen i første gir
        GirNivaa = MIN_GIR;
    } //if (GirNivaa == REVERS)
} //if (nyTast.IsKeyDown(Keys.Up))
#endregion

#region FLYTT BIL BAKOVER
//skal bilen forflyttes bakover?
if (nyTast.IsKeyDown(Keys.Down) && !forrigeTast.IsKeyDown(Keys.Down)) {
    forflytning = +1;
    GirNivaa = REVERS;
} //if (nyTast.IsKeyDown(Keys.Down))
//skal bilen forflyttes bakover?
if (nyTast.IsKeyDown(Keys.S) && !forrigeTast.IsKeyDown(Keys.S)) {
    forflytning = +1;
    GirNivaa = REVERS;
} //if (nyTast.IsKeyDown(Keys.Down))
#endregion

//skal bilen forflyttes til høyre?
if (nyTast.IsKeyDown(Keys.Right) || nyTast.IsKeyDown(Keys.D)) {
    venstrehoyreRotasjon = -0.05f;
} //if (nyTast.IsKeyDown(Keys.Right))

//skal bilen forflyttes til venstre?
if (nyTast.IsKeyDown(Keys.Left) || nyTast.IsKeyDown(Keys.A)) {
    venstrehoyreRotasjon = 0.05f;
} //if (nyTast.IsKeyDown(Keys.Left))

//zooming
if (nyTast.IsKeyDown(Keys.Z) && !forrigeTast.IsKeyDown(Keys.Z)) {
    Synsvinkel++;
} //if (nyTast.IsKeyDown(Keys.Left))

#region GIR OPP/NED
if (nyTast.IsKeyDown(Keys.Q) && !forrigeTast.IsKeyDown(Keys.Q)) {
    //er gjeldende gir lavere enn høyeste gir?
    if (GirNivaa < MAX_GIR && GirNivaa != REVERS) {
        GirNivaa++;
    } //if (GirNivaa < MAX_GIR)
} //if (nyTast.IsKeyDown(Keys.Q))

```

```

if (nyTast.IsKeyDown(Keys.E) && !forrigeTast.IsKeyDown(Keys.E)) {
    //er gjeldende gir høyere enn laveste gir?
    if (GirNivaa > MIN_GIR && GirNivaa != REVERS) {
        GirNivaa--;
    } //if (GirNivaa > MIN_GIR)
} //if (nyTast.IsKeyDown(Keys.Left))
#endregion

#region HJELP
//vil bruker vise/skjule hjelp?
if (nyTast.IsKeyDown(Keys.F1) && !forrigeTast.IsKeyDown(Keys.F1)) {
    VisHjelp = !VisHjelp;
} //if (nyTast.IsKeyDown(Keys.F1))
//vil bruker vise/skjule hjelp?
if (nyTast.IsKeyDown(Keys.H) && !forrigeTast.IsKeyDown(Keys.H)) {
    VisHjelp = !VisHjelp;
} //if (nyTast.IsKeyDown(Keys.H))
#endregion

//vil bruker starte et nytt spill?
if (nyTast.IsKeyDown(Keys.R) && !forrigeTast.IsKeyDown(Keys.R)) {
    ResetSpill();
} //if (nyTast.IsKeyDown(Keys.R))

//sett forrige tast til tasten som ble trykket nå
forrigeTast = nyTast;
//oppdater posisjonen til kamera
OppdaterPosisjon(new Vector3(0, 0, forflytning), venstrehoyreRotasjon);
} //UpdateInput

/// <summary>
/// Oppdaterer bilens posisjon basert på knapp som ble trykket
/// </summary>
/// <param name="nyPosisjon">Den nye posisjonen: Vector3</param>
/// <param name="venstrehoyreRotasjon">rotasjon på x-aksen (mot høyre eller
///                                     mot venstre): float</param>
private void OppdaterPosisjon(Vector3 nyPosisjon, float venstrehoyreRotasjon)
{
    //siden spiller kun skal flytte bil langs x - og z-akse er denne alltid
    // null skulle spiller kunne forflytte seg oppover måtte denne vært med i
    // if-test for Keys.Up/W og Keys.Down/S
    float oppnedRotasjon = 0.0f;
    //beregner en rotasjon som går opp eller ned ved bruk av Quaternion
    Quaternion oppnedRotQuat = Quaternion.CreateFromAxisAngle(new Vector3(1,
                                                                    0, 0), oppnedRotasjon);
    //beregner en rotasjon som går til høyre eller venstre ved bruk av
    Quaternion
    Quaternion vhRotQuat = Quaternion.CreateFromAxisAngle(new Vector3(0, 1,
                                                                    0), venstrehoyreRotasjon);
    //finn tilleggsrotasjonen ved å gange rotasjonen rundt aksene med
    hverandre (rekkefølge likegyldig)
    Quaternion tilleggsRotasjon = oppnedRotQuat * vhRotQuat;
    //beregner kameraets rotasjon ved å multiplisere kameraets nåværende
    posisjon med tilleggsrotasjonen
    //Quaternion er ikke som vanlig matrisemultiplikasjon, så her har
    rekkefølgen betydning
    //(den var likegyldig over pga aksene er loddrette ovenfor hverandre)
    //her vil tilleggsRotasjon først bli rotert rundt aksene til kameraRotasjon
    kameraRotasjon = kameraRotasjon * tilleggsRotasjon;
    //transformerer forflytningen med rotasjonen rundt aksene for å lage ny
    vektor
    Vector3 rotertVektor = Vector3.Transform(nyPosisjon, kameraRotasjon);
}

```



```

        //oppdaterer posisjonen
        posisjon += Fart * rotertVektor;
        OppdaterViewMatrix();
    } //oppdaterPosisjon

    private void OppdaterViewMatrix() {
        //sett originalverdier
        Vector3 originalZ = new Vector3(0, 0, 1);
        Vector3 originalOppVektor = new Vector3(0, 1, 0);
        //transformer rotasjon om z-aksen
        Vector3 kameraRotertZ = Vector3.Transform(originalZ, kameraRotasjon);
        Vector3 endeligZ = posisjon + kameraRotertZ;
        //transformer rotasjon om y-aksen
        Vector3 kameraRotertY = Vector3.Transform(originalOppVektor,
kameraRotasjon);
        //oppdater view matrisen
        viewMatrix = Matrix.CreateLookAt(endeligZ, posisjon, kameraRotertY);
    } //oppdaterViewMatrix

    private void KrasjerMedSkybox() {
        //hent ut kameraets gjeldende posisjon
        float x = Posisjon.X,
            y = Posisjon.Y,
            z = Posisjon.Z;
        float beregnNyPos = 0.0f;
        //opprett array som inneholder posisjon for veggene til skybox
        //og objekter som er inkludert i skybox
        float[,] krasjArray = {
            {
                //x-posisjon vegger skybox
                -42.2f, //0
                43.5f, //1
                //x-posisjon garasje (i skybox)
                -34.6f, //2
                -40.3f, //3
                //x-posisjon bensinstasjon (i skybox)
                8.7f, //4
                10.0f, //5
            },
            {
                //z-posisjon vegger skybox
                -41.5f, //0
                44.6f, //1
                //z-posisjon garasje (i skybox)
                -37.0f, //2
                -40.3f, //3
                //z-posisjon bensinstasjon (i skybox)
                -40.2f, //4
                -41.2f //5
            }
        };
    }

```

```

#region IF TESTER GARASJE
//er kamera innenfor området til garasjen (hjørnekoordinater)
if (x <= krasjArray[0, 2] && x >= krasjArray[0, 3] && z <=
    krasjArray[1, 2] && z >= krasjArray[1, 3]) {
    //for å skille mellom veggene og unngå at man går gjennom veggen eller
    // plutselig havner på siden av en annen vegg, så måtte jeg lage
    //begrensninger som legges til/trekkes fra hjørne
    float frontSide = krasjArray[1, 3] + 2.0f; //ta koordinat for bakvegg
    (z) og legg øk verdi
    float bakSide = krasjArray[1, 3]; //koordinat for bakside (z)
    float hoyreside = krasjArray[0, 2] - 1.0f; //ta koordinat for høyre
    hjørne (x) og legg til
    float venstreSide = krasjArray[0, 3] + 1.0f; //ta koordinat for
    venstre hjørne (x) og trekk fra
    //er kamera innenfor høyre garasjevegg?
    if (x >= hoyreside && z <= krasjArray[1, 2] && z >= krasjArray[1, 3])
    {
        //sett kameraets posisjon på x-aksen
        x = krasjArray[0, 2];
        //beregn verdien til hvor på z-aksen kameraet var, ved å ta
        //veggen z-verdi og legge til differansen
        //eksempel: ved krasj er z = 38; da blir beregnNyPos = 38 - 37 = 1
        //dette gir da z = 37 + 1 = 38, altså punktet hvor kameraet
        kolliderte med veggen
        beregnNyPos = z - krasjArray[1, 2];
        z = krasjArray[1, 2] + beregnNyPos;
    } else if (x <= venstreSide && z <= krasjArray[1, 2] && z >=
        krasjArray[1, 3]) { //innenfor venstre vegg?
        //sett kameraets posisjon på x-aksen
        x = krasjArray[0, 3];
        //beregn hvor kameraet kolliderte med veggen
        beregnNyPos = z - krasjArray[1, 2];
        z = krasjArray[1, 2] + beregnNyPos;
    } else if (x <= krasjArray[0, 2] && x >= krasjArray[0, 3] && z >=
        frontSide) { //innenfor frontveggen?
        //sett kameraets posisjon på z-aksen
        z = krasjArray[1, 2];
        //beregn hvor kameraet kolliderte med veggen
        beregnNyPos = x - krasjArray[0, 2];
        x = krasjArray[0, 2] + beregnNyPos;
    } else if (x <= krasjArray[0, 2] && x >= krasjArray[0, 3] && z >=
        bakSide) { //innenfor bakveggen?
        //sett kameraets posisjon på z-aksen
        z = krasjArray[1, 3];
        //beregn hvor kameraet kolliderte med veggen
        beregnNyPos = x - krasjArray[0, 2];
        x = krasjArray[0, 2] + beregnNyPos;
    } //if (x >= hoyreside ...)
} //if (x <= krasjArray[0, 2]) ...)
#endregion

```

```

        #region IF TESTER SKYBOX
        //krasjer kamera med en vegg/et objekt?
        if (x <= krasjArray[0, 0]) {
            //hent ut koordinatene til veggen/objektet/objektet
            x = krasjArray[0, 0];
        } //if (x <= krasjArray[0, 0])
        if (x >= krasjArray[0, 1]) {
            //hent ut koordinatene til veggen/objektet
            x = krasjArray[0, 1];
        } //if (x >= krasjArray[0, 1])
        if (z <= krasjArray[1, 0]) {
            //hent ut koordinatene til veggen/objektet
            z = krasjArray[1, 0];
        } //if (z <= krasjArray[1, 0])
        if (z >= krasjArray[1, 1]) {
            //hent ut koordinatene til veggen/objektet
            z = krasjArray[1, 1];
        } //if (z >= krasjArray[1, 1])
        //sett (ny) kamera posisjon
        Posisjon = new Vector3(x, y, z);
        #endregion
    } //krasjerMedSkybox
#endregion

#region GET & SET METODER
/// <summary>
/// Returnerer kameraets gjeldene posisjon (Vector3)
/// </summary>
internal static Vector3 Posisjon {
    get {
        return posisjon;
    }
    set {
        posisjon = value;
    }
}

/// <summary>
/// Returnerer kameraets gjeldende rotasjon (Quaternion)
/// </summary>
internal static Quaternion Rotasjon {
    get {
        return kameraRotasjon;
    }
}

/// <summary>
/// Hent eller sett verdien for ViewAngle
/// (brukes til oppretting av ProjectionMatrix)
/// </summary>
public float ViewAngle {
    get {
        return viewAngle;
    }
    set {
        viewAngle = value;
    }
}
}

```

```

/// <summary>
/// Hent eller sett verdien for AspectRatio
/// (brukes til oppretting av ProjectionMatrix)
/// </summary>
public float AspectRatio {
    get {
        return aspectRatio;
    }
    set {
        aspectRatio = value;
    }
}

/// <summary>
/// Hent eller sett verdien for NearPlane
/// (brukes til oppretting av ProjectionMatrix)
/// </summary>
public float NearPlane {
    get {
        return nearPlane;
    }
    set {
        nearPlane = value;
    }
}

/// <summary>
/// Hent eller sett verdien for FarPlane
/// (brukes til oppretting av ProjectionMatrix)
/// </summary>
public float FarPlane {
    get {
        return farPlane;
    }
    set {
        farPlane = value;
    }
}

/// <summary>
/// Returnerer view matrisen
/// </summary>
internal static Matrix ViewMatrix {
    get {
        return viewMatrix;
    }
}

/// <summary>
/// Returnerer projection matrisen
/// </summary>
internal static Matrix ProjectionMatrix {
    get {
        return projectionMatrix;
    }
}

```

```

/// <summary>
/// Returnerer aktivt gir
/// </summary>
internal static int GirNivaa {
    get {
        return girNivaa;
    }
    private set {
        girNivaa = value;
    }
}

/// <summary>
/// Returnerer farten basert på hvilket gir som er aktivt.
/// Farten starter på 0.05f og hastighet = fart * gir
/// Eksempelvis: 3. gir => 0.05f * 3 = 0.15f
/// Farten ved reversering er 0.05f
/// </summary>
private float Fart {
    get {
        float fart = 0.05f;
        switch (GirNivaa) {
            case MIN_GIR:
                fart = 0.05f;
                break;
            case 2:
                fart = 0.10f;
                break;
            case 3:
                fart = 0.15f;
                break;
            case MAX_GIR:
                fart = 0.20f;
                break;
            case REVERS:
                fart = 0.05f;
                break;
        } //switch
        return fart;
    } //get
}

/// <summary>
/// Returnerer true/false basert på om bruker vil se/skjule Hjelp
/// </summary>
internal static bool VisHjelp {
    get {
        return visHjelp;
    }
    //private slik at den ikke blir overstyrt
    private set {
        visHjelp = value;
    }
}

```

```

/// <summary>
/// Returnerer synsvinkelen (zoom)
/// Det er to visninger; bak bilen og
/// frontvindu (ser ikke bilen)
/// </summary>
internal static int Synsvinkel {
    get {
        return zoom;
    }
    private set {
        if (zoom < MAX_ZOOM) {
            zoom = value;
        } else {
            zoom = 1;
        } //if (zoom < MAX_ZOOM)
    } //set
}

/// <summary>
/// Returner boolsk verdi basert på om bruker ønsker å
/// starte et nytt spill. Hvis nytt spill SKAL startes,
/// sett verdi til true. Hvis nytt spill HAR startet,
/// sett verdi til false.
/// </summary>
internal static bool StartNyttSpill {
    get {
        return nyttSpill;
    }
    set {
        nyttSpill = value;
    }
}
#endregion
} //QuaternionKamera

```

XNAUtils

```
/// <summary>
/// Klasse som oppretter BoundingBox og BoundingSphere
/// Tatt fra XNA Game Programming Recipes 3.0 av Riemer, s. 102
/// </summary>
internal static class XNAUtils {
    internal static BoundingBox TransformBoundingBox(BoundingBox origBox, Matrix
matrix) {
        Vector3 origCorner1 = origBox.Min;
        Vector3 origCorner2 = origBox.Max;

        Vector3 transCorner1 = Vector3.Transform(origCorner1, matrix);
        Vector3 transCorner2 = Vector3.Transform(origCorner2, matrix);

        return new BoundingBox(transCorner1, transCorner2);
    }

    internal static BoundingSphere TransformBoundingSphere(BoundingSphere
originalBoundingSphere, Matrix transformationMatrix) {
        Vector3 trans;
        Vector3 scaling;
        Quaternion rot;
        transformationMatrix.Decompose(out scaling, out rot, out trans);

        float maxScale = scaling.X;
        if (maxScale < scaling.Y)
            maxScale = scaling.Y;
        if (maxScale < scaling.Z)
            maxScale = scaling.Z;

        float transformedSphereRadius = originalBoundingSphere.Radius * maxScale;
        Vector3 transformedSphereCenter =
Vector3.Transform(originalBoundingSphere.Center, transformationMatrix);

        BoundingSphere transformedBoundingSphere = new
BoundingSphere(transformedSphereCenter, transformedSphereRadius);

        return transformedBoundingSphere;
    }

    internal static Model LoadModelWithBoundingSphere(ref Matrix[]
modelTransforms, string asset, ContentManager content) {
        Model newModel = content.Load<Model>(asset);

        modelTransforms = new Matrix[newModel.Bones.Count];
        newModel.CopyAbsoluteBoneTransformsTo(modelTransforms);

        BoundingSphere completeBoundingSphere = new BoundingSphere();
        foreach (ModelMesh mesh in newModel.Meshes) {
            BoundingSphere origMeshSphere = mesh.BoundingSphere;
            BoundingSphere transMeshSphere =
XNAUtils.TransformBoundingSphere(origMeshSphere,
modelTransforms[mesh.ParentBone.Index]);
            completeBoundingSphere =
BoundingSphere.CreateMerged(completeBoundingSphere, transMeshSphere);
        }
        newModel.Tag = completeBoundingSphere;

        return newModel;
    }
}
```

```

        internal static void DrawBoundingBox(BoundingBox bBox, GraphicsDevice device,
        BasicEffect basicEffect, Matrix worldMatrix, Matrix viewMatrix, Matrix
        projectionMatrix) {
            Vector3 v1 = bBox.Min;
            Vector3 v2 = bBox.Max;

            VertexPositionColor[] cubeLineVertices = new VertexPositionColor[8];
            cubeLineVertices[0] = new VertexPositionColor(v1, Color.White);
            cubeLineVertices[1] = new VertexPositionColor(new Vector3(v2.X, v1.Y,
            v1.Z), Color.Red);
            cubeLineVertices[2] = new VertexPositionColor(new Vector3(v2.X, v1.Y,
            v2.Z), Color.Green);
            cubeLineVertices[3] = new VertexPositionColor(new Vector3(v1.X, v1.Y,
            v2.Z), Color.Blue);
            cubeLineVertices[4] = new VertexPositionColor(new Vector3(v1.X, v2.Y,
            v1.Z), Color.White);
            cubeLineVertices[5] = new VertexPositionColor(new Vector3(v2.X, v2.Y,
            v1.Z), Color.Red);
            cubeLineVertices[6] = new VertexPositionColor(v2, Color.Green);
            cubeLineVertices[7] = new VertexPositionColor(new Vector3(v1.X, v2.Y,
            v2.Z), Color.Blue);

            short[] cubeLineIndices = { 0, 1, 1, 2, 2, 3, 3, 0, 4, 5, 5, 6, 6, 7, 7,
            4, 0, 4, 1, 5, 2, 6, 3, 7 };

            basicEffect.World = worldMatrix;
            basicEffect.View = viewMatrix;
            basicEffect.Projection = projectionMatrix;
            basicEffect.VertexColorEnabled = true;

            RasterizerState rasterizerState1 = new RasterizerState();
            rasterizerState1.CullMode = CullMode.None;
            rasterizerState1.FillMode = FillMode.Solid;

            foreach (EffectPass pass in basicEffect.CurrentTechnique.Passes) {
                pass.Apply();
            }

            device.DrawUserIndexedPrimitives<VertexPositionColor>(PrimitiveType.LineList,
            cubeLineVertices, 0, 8, cubeLineIndices, 0, 12);
        }
    }
}

```



```

internal static void DrawSphereSpikes(BoundingSphere sphere, GraphicsDevice
device, BasicEffect basicEffect, Matrix worldMatrix, Matrix viewMatrix, Matrix
projectionMatrix) {
    Vector3 up = sphere.Center + sphere.Radius * Vector3.Up;
    Vector3 down = sphere.Center + sphere.Radius * Vector3.Down;
    Vector3 right = sphere.Center + sphere.Radius * Vector3.Right;
    Vector3 left = sphere.Center + sphere.Radius * Vector3.Left;
    Vector3 forward = sphere.Center + sphere.Radius * Vector3.Forward;
    Vector3 back = sphere.Center + sphere.Radius * Vector3.Backward;

    VertexPositionColor[] sphereLineVertices = new VertexPositionColor[6];
    sphereLineVertices[0] = new VertexPositionColor(up, Color.White);
    sphereLineVertices[1] = new VertexPositionColor(down, Color.White);
    sphereLineVertices[2] = new VertexPositionColor(left, Color.White);
    sphereLineVertices[3] = new VertexPositionColor(right, Color.White);
    sphereLineVertices[4] = new VertexPositionColor(forward, Color.White);
    sphereLineVertices[5] = new VertexPositionColor(back, Color.White);

    basicEffect.World = worldMatrix;
    basicEffect.View = viewMatrix;
    basicEffect.Projection = projectionMatrix;
    basicEffect.VertexColorEnabled = true;
    foreach (EffectPass pass in basicEffect.CurrentTechnique.Passes) {
        pass.Apply();
        device.DrawUserPrimitives<VertexPositionColor>(PrimitiveType.LineList,
sphereLineVertices, 0, 3);
    }
}

internal static VertexPositionColor[] VerticesFromVector3List(List<Vector3>
pointList, Color color) {
    VertexPositionColor[] vertices = new VertexPositionColor[pointList.Count];

    int i = 0;
    foreach (Vector3 p in pointList)
        vertices[i++] = new VertexPositionColor(p, color);

    return vertices;
}

internal static BoundingBox CreateBoxFromSphere(BoundingSphere sphere) {
    float radius = sphere.Radius;
    Vector3 outerPoint = new Vector3(radius, radius, radius);

    Vector3 p1 = sphere.Center + outerPoint;
    Vector3 p2 = sphere.Center - outerPoint;

    return new BoundingBox(p1, p2);
}
} //XNAUtils

```