Otto-von-Guericke-Universität Magdeburg
Arbeitsgruppe Data & Knowledge Engineering
M. Hinz, T. Petrachi, M. Saathoff, E. Schondorff, F. Uhde

Entwicklerdokumentation

Inhaltsverzeichnis

#

Übersicht über das SDK

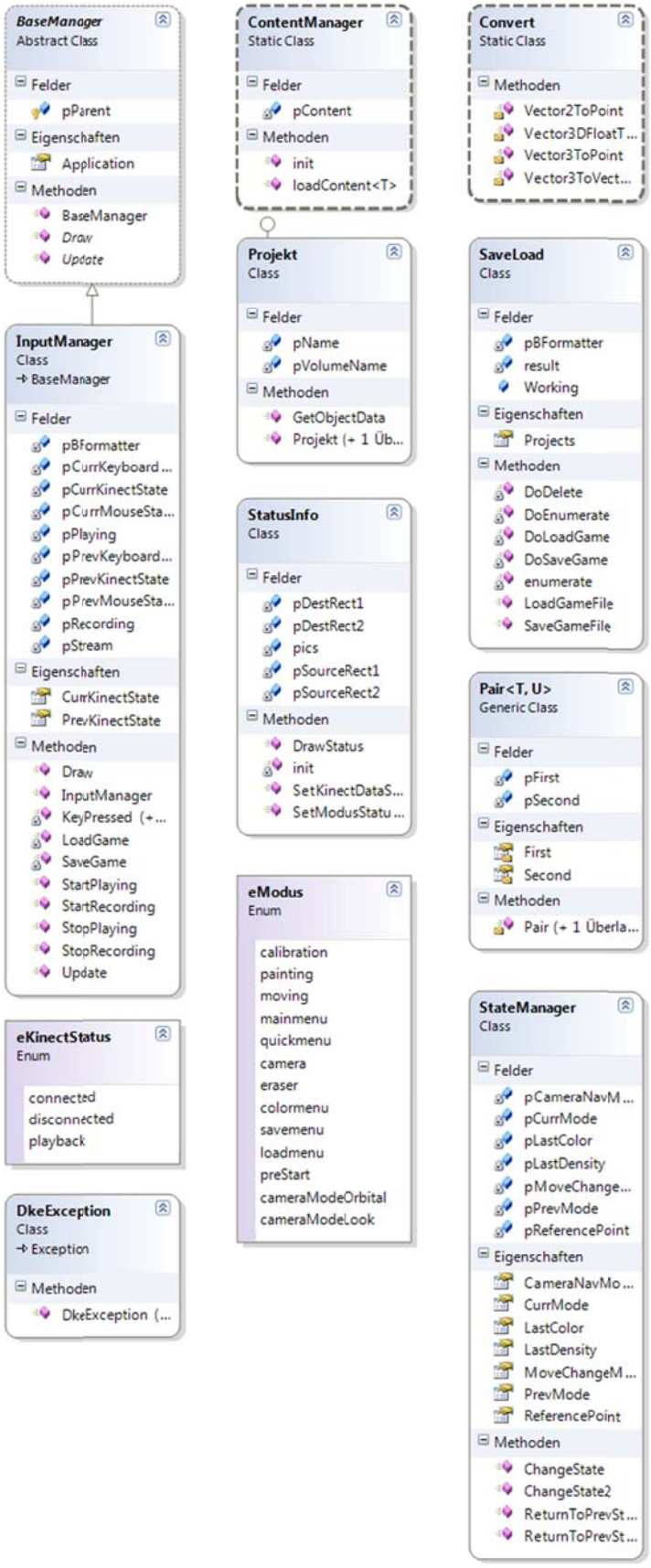
Das DKE_Kinect SDK stellt ein grafisches Zeichenprogramm mit der XBOX360 Kinect als Eingabegerät zur Verfügung.
Entwicklungstechnisch ist es in 4 Bereiche unterteilt:

- Core
- Grafik
- Kinect
- PolyVox

Diese Entwicklerdoku soll die Funktionen und Aufgaben der einzelnen Klassen erläutern um so eine Weiterentwicklung zu ermöglichen.

Dem SDK liegt als Programmiersprache *C#* (*C – Sharp*) zugrunde. Weiterhin wurde das *„XNA – Framework“* sowie das *„Kincet for Windows SDK from Microsoft Research“* genutzt.

Core


BaseManager
Abstract Class
Felder
pParent
Eigenschaften
Application
Methoden
BaseManager
Draw
Update
ContentManager
Static Class
Felder
pContent
Methoden
init
loadContent<T>
Convert
Static Class
Methoden
Vector2ToPoint
Vector3DFloatT...
Vector3ToPoint
Vector3ToVect...
Projekt
Class
Felder
pName
pVolumeName
Methoden
GetObjectData
Projekt (+ 1 Üb...
SaveLoad
Class
Felder
pBFormatter
result
Working
Eigenschaften
Projects
Methoden
DoDelete
DoEnumerate
DoLoadGame
DoSaveGame
enumerate
LoadGameFile
SaveGameFile
InputManager
Class
BaseManager
Felder
pBFormatter
pCurrKeyboard...
pCurrKinectState
pCurrMouseSta...
pPlaying
pPrevKeyboard...
pPrevKinectState
pPrevMouseSta...
pRecording
pStream
Eigenschaften
CurrKinectState
PrevKinectState
Methoden
Draw
InputManager
KeyPressed (+...
LoadGame
SaveGame
StartPlaying
StartRecording
StopPlaying
StopRecording
Update
StatusInfo
Class
Felder
pDestRect1
pDestRect2
pics
pSourceRect1
pSourceRect2
Methoden
DrawStatus
init
SetKinectDataS...
SetModusStatu...
Pair<T, U>
Generic Class
Felder
pFirst
pSecond
Eigenschaften
First
Second
Methoden
Pair (+ 1 Überla...
eKinectStatus
Enum
connected
disconnected
playback
eModus
Enum
calibration
painting
moving
mainmenu
quickmenu
camera
eraser
colormenu
savemenu
loadmenu
preStart
cameraModeOrbital
cameraModeLook
StateManager
Class
Felder
pCameraNavM...
pCurrMode
pLastColor
pLastDensity
pMoveChange...
pPrevMode
pReferencePoint
Eigenschaften
CameraNavMo...
CurrMode
LastColor
LastDensity
MoveChangeM...
PrevMode
ReferencePoint
Methoden
ChangeState
ChangeState2
ReturnToPrevSt...
ReturnToPrevSt...
DkeException
Class
Exception
Methoden
DkeException (...

Im Namespace `DKE_Kinect.Entwicklung.Kinect` befinden sich alle Klassen, die für die allgemeine Verwaltung der Application verantwortlich sind. Hier ist die abstrakte Klasse `BaseManager` definiert, von der alle anderen Programm Manager erben. Sie stellt eine Update und Draw Routine zur Verfügung und hält eine Referenz auf die verwaltende Applikation.
Der `ContentManager` kapselt den Content Manager von XNA, und bietet als statische Klasse Zugriff auf Asset Lade Funktionen von jeden Punkt des Programms aus.
Eine weitere Hilfsklasse ist die `DkeException` Klasse, welche es erlaubt eigene Fehlermeldungen zu erstellen und vorhandene Fehler zu kapseln und mit eigenen Hinweisen zu versehen.
Der `StateManager` ist verantwortlich für den Status unsere Applikation, er verwendet die Enum `eKinectStatus` und `eModus` um die Datenquelle (*connected*, *disconnected*, oder *playback*) und den aktuellen Programmstatus zu kodieren. Über diesen Manager werden auch Status Änderungen initiiert und der Programmfluss gelenkt.
Die Klasse `StatusInfo` stellt ein kleines grafisches Interface zur Verfügung, mit dem der Datenquellenstatus und Programmstatus visualisiert werden kann.

Eine weitere Kernklasse ist der InputManager. Dieser Manager verwaltet jeglcihen Input der Applikation und speichert sowohl den aktuellen, als auch den vorherigen, Status des Keyboards, der Maus und der Kinect. Über diesen Manager können neue Events gehooked, Routinen auf neue Keys gemappt und Aufzeichnungen angelegt oder geladen werden.

Die Klassen `Convert` und `Pair` sind Hilfsklassen für die Menü Struktur und für die Kommunikation mit der C++ Library von PolyVox.
Als letztes gibt es noch die `Projekt` und `Save/Load` Klasse. `Projekt` ist noch sehr rudimentär und enthält bisher nur einen Namen und einen Bezeichner für ein Volumen. Über die `Save/Load` Klasse können Volumina in dem für XNA Spiel reservierten Container unter */User/Dokumente/MySavedGames/DKE_Kinect/* gespeichert werden und auch von dort wieder geladen werden.

Erweiterungsmöglichkeiten

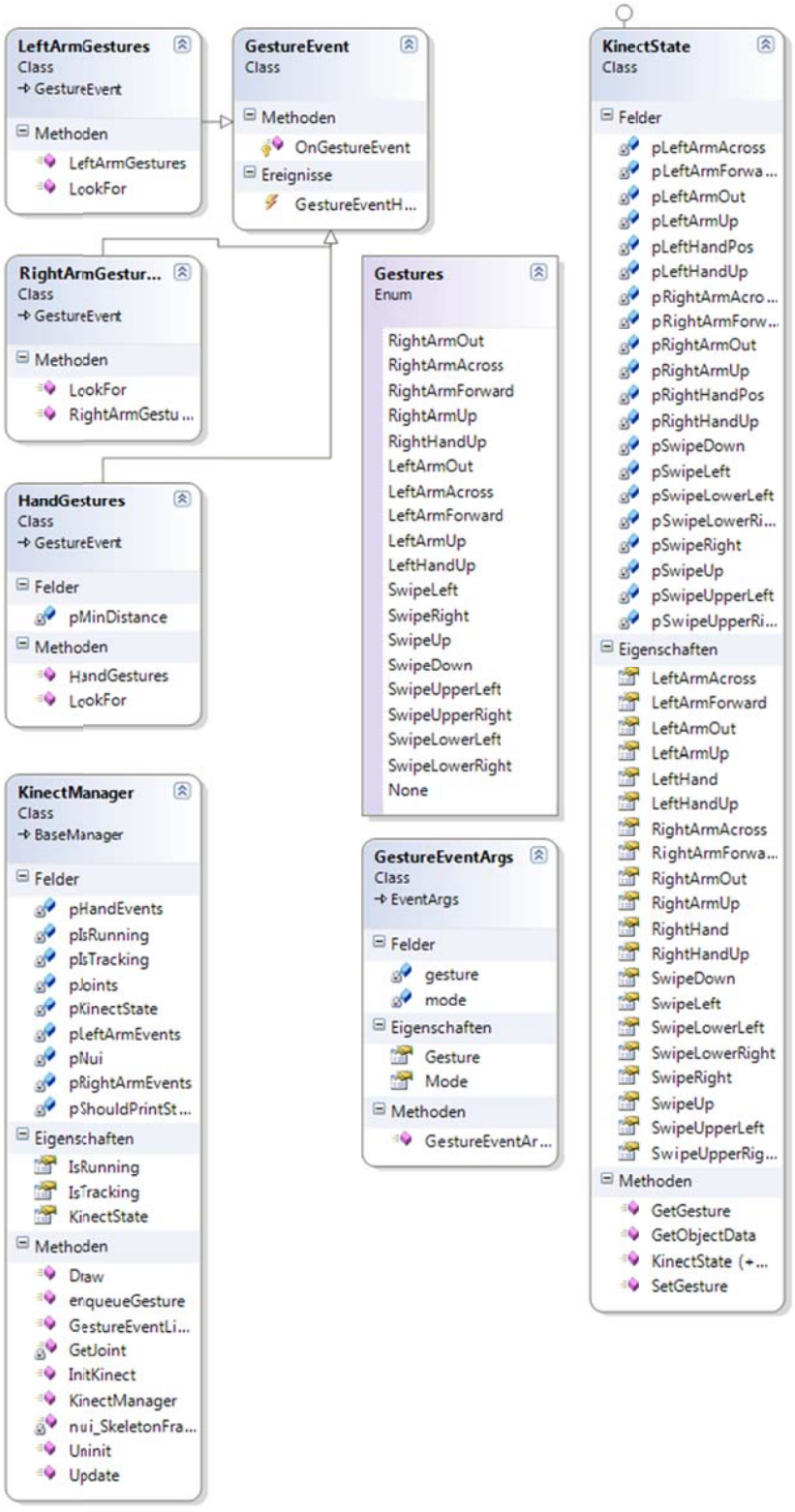
Projekt - Klasse ausbauen

Die Projektklasse kann noch erweitert werden, um z.B. ein Thumbnail des Objekts, oder eine eigene Palette mit zu speichern. Hierzu müsste die Projektklasse sowie Save/Load angepasst werden.

Tastatur/Maus Steuerung

Durch Anpassung des Inputmanagers könnte die Steuerung von Painect auf Tastatur und Maus gemappt werden, um zum Beispiel eigenen Modifikationen ohne Kinect testen zu können.

Kinect


LeftArmGestures
Class
GestureEvent
Methoden
LeftArmGestures
LookFor
GestureEvent
Class
Methoden
OnGestureEvent
Ereignisse
GestureEventH...
KinectState
Class
Felder
pLeftArmAcross
pLeftArmForwa...
pLeftArmOut
pLeftArmUp
pLeftHandPos
pLeftHandUp
pRightArmAcro...
pRightArmForw...
pRightArmOut
pRightArmUp
pRightHandPos
pRightHandUp
pSwipeDown
pSwipeLeft
pSwipeLowerLeft
pSwipeLowerRi...
pSwipeRight
pSwipeUp
pSwipeUpperLeft
pSwipeUpperRi...
Eigenschaften
LeftArmAcross
LeftArmForward
LeftArmOut
LeftArmUp
LeftHand
LeftHandUp
RightArmAcross
RightArmForwa...
RightArmOut
RightArmUp
RightHand
RightHandUp
SwipeDown
SwipeLeft
SwipeLowerLeft
SwipeLowerRight
SwipeRight
SwipeUp
SwipeUpperLeft
SwipeUpperRig...
Methoden
GetGesture
GetObjectData
KinectState (+...
SetGesture
RightArmGestur...
Class
GestureEvent
Methoden
LookFor
RightArmGestu...
Gestures
Enum
RightArmOut
RightArmAcross
RightArmForward
RightArmUp
RightHandUp
LeftArmOut
LeftArmAcross
LeftArmForward
LeftArmUp
LeftHandUp
SwipeLeft
SwipeRight
SwipeUp
SwipeDown
SwipeUpperLeft
SwipeUpperRight
SwipeLowerLeft
SwipeLowerRight
None
HandGestures
Class
GestureEvent
Felder
pMinDistance
Methoden
HandGestures
LookFor
GestureEventArgs
Class
EventArgs
Felder
gesture
mode
Eigenschaften
Gesture
Mode
Methoden
GestureEventAr...
KinectManager
Class
BaseManager
Felder
pHandEvents
pIsRunning
pIsTracking
pJoints
pKinectState
pLeftArmEvents
pNui
pRightArmEvents
pShouldPrintSt...
Eigenschaften
IsRunning
IsTracking
KinectState
Methoden
Draw
enqueueGesture
GestureEventLi...
GetJoint
InitKinect
KinectManager
nui_SkeletonFra...
Uninit
Update

Im Namespace `DKE_Kinect.Entwicklung.Kinect` sind die für das Einlesen der Kinect-Daten relevanten Klassen zu finden.
Das Einlesen/Abholen der Kinect-Daten übernimmt der `KinectManager`. Dabei wird die Kinect mit Hilfe der Funktionen aus dem Kinect SDK initialisiert. Anschließend können die Skelett-Daten Frame-weise über den entsprechenden Handler abgeholt und verarbeitet werden:

```
pNui.SkeletonFrameReady +=
        new EventHandler<SkeletonFrameReadyEventArgs>(nui_SkeletonFrameReady);
```

Das Event `GestureEvent` bildet die Grundlage für die Erkennung von Gesten. Die davon abgeleiteten Klassen `LeftArmGestures`, `RightArmGestures`, `HandGestures` übernehmen dabei die Erkennung von Gesten der jeweiligen Körperteile. Wird eine zulässige Geste erkannt, so wird ein entsprechendes Event mit der jeweiligen Geste (dem Enum `Gestures` zu entnehmen) und dem aktuellen Modus des Programms geworfen, z.B.:

```
OnGestureEvent(new GestureEventArgs(
        Gestures.RightHandUp, StateManager.CurrMode));
```

Im Update-Zyklus des `KinectManagers` werden die aktuellen Positionen des Skeletts (z.Z. nur die Hände) sowie die geworfenen Gesten des aktuellen Frames in einer eigenen Struktur – dem `KinectState` gespeichert. Diese Struktur wird in Form einer Klasse, die alle Gesten als boolesche Variable sowie die Handpositionen als Vektoren, bereitgestellt. Somit können die Kinect-Daten im weiteren Programmablauf verwendet werden.

Erweiterungsmöglichkeiten

Weitere Gesten definieren/implementieren

Es ist problemlos möglich, weitere Gesten zu implementieren. Dazu müssen nur einige wenige Voraussetzungen beachtet werden:

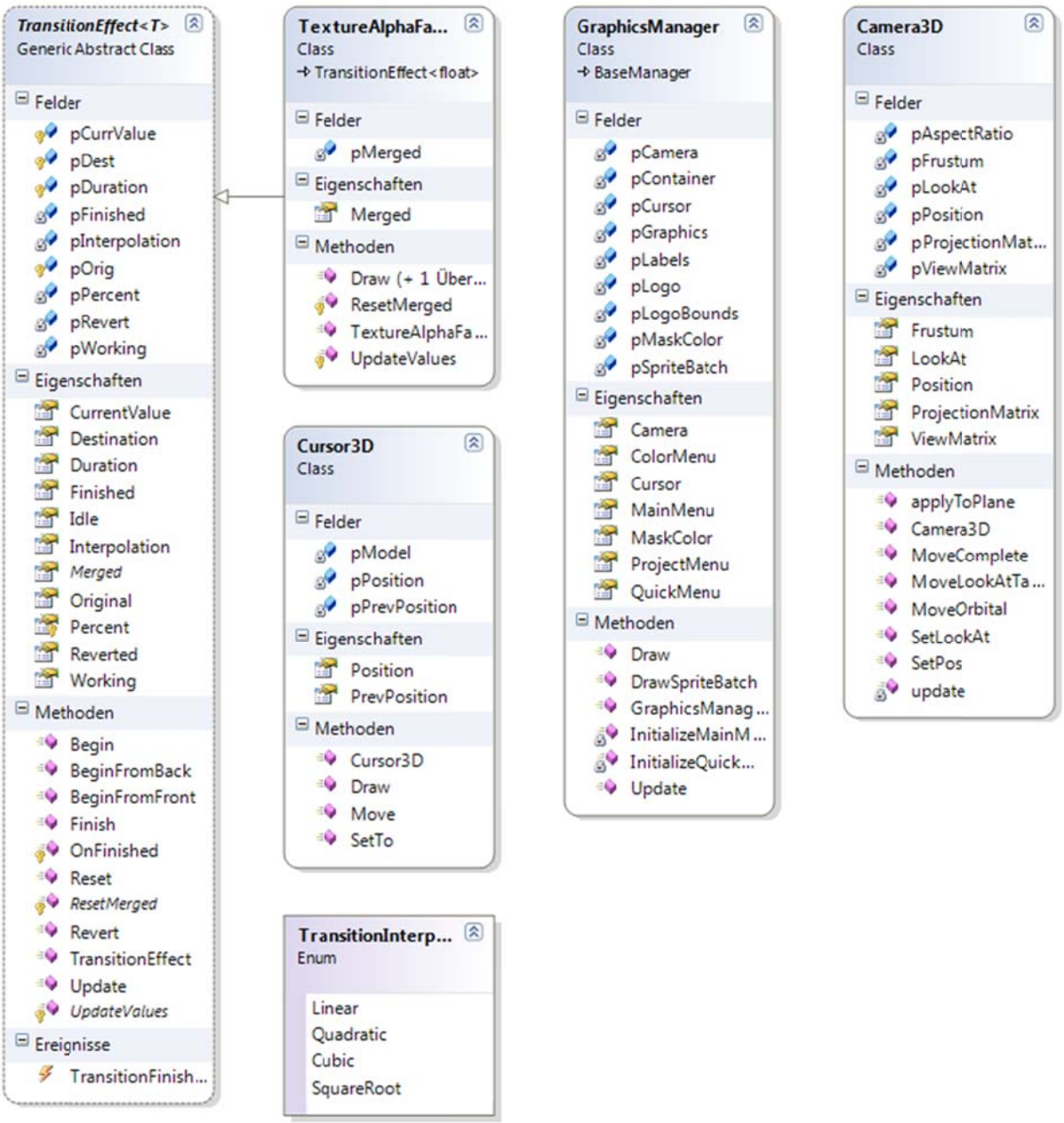
- Es besteht die Möglichkeit, eine neue `GestureEvent`-Klasse zu erstellen (siehe `LeftArmGestures`, `RightArmGestures`, `HandGestures`) oder die die bestehenden Klassen zu erweitern.
- Dabei müssen alle neuen Gesten in das `Gestures`-Enum aufgenommen werden und zu jeder Geste das entsprechende Datenfeld im `KinectState` angelegt werden.
- Zudem muss sichergestellt werden, dass der `GestureEventHandler` die Gesten im gewünschten Modus auch zulässt. Dazu müssen die Gesten im entsprechenden *case*-Block der *enqueueGesture*-Methode (`KinectManager`) hinzugefügt werden.

Weitere Skelett-Knoten auslesen

Es werden bereits die wichtigsten Skelett-Knoten im Handler des `SkeletonFrameReady`-Events ausgelesen, allerdings werden nur die Hände in den `KinectState` übernommen. Weitere Punkte können analog dazu dem `KinectState` übergeben werden. Dazu muss lediglich ein Feld für den jeweiligen Vektor erstellt und diesem die Koordinaten aus dem Joints-Dicitonary des `KinectManagers` übergeben werden:

```
this.pKinectState.RightHand = this.pJoints[JointID.HandRight];
```

Grafik


TransitionEffect<T>
Generic Abstract Class
Felder
pCurrValue
pDest
pDuration
pFinished
pInterpolation
pOrig
pPercent
pRevert
pWorking
Eigenschaften
CurrentValue
Destination
Duration
Finished
Idle
Interpolation
Merged
Original
Percent
Reverted
Working
Methoden
Begin
BeginFromBack
BeginFromFront
Finish
OnFinished
Reset
ResetMerged
Revert
TransitionEffect
Update
UpdateValues
Ereignisse
TransitionFinish...
TextureAlphaFa...
Class
TransitionEffect<float>
Felder
pMerged
Eigenschaften
Merged
Methoden
Draw (+ 1 Über...
ResetMerged
TextureAlphaFa...
UpdateValues
Cursor3D
Class
Felder
pModel
pPosition
pPrevPosition
Eigenschaften
Position
PrevPosition
Methoden
Cursor3D
Draw
Move
SetTo
TransitionInterp...
Enum
Linear
Quadratic
Cubic
SquareRoot
GraphicsManager
Class
BaseManager
Felder
pCamera
pContainer
pCursor
pGraphics
pLabels
pLogo
pLogoBounds
pMaskColor
pSpriteBatch
Eigenschaften
Camera
ColorMenu
Cursor
MainMenu
MaskColor
ProjectMenu
QuickMenu
Methoden
Draw
DrawSpriteBatch
GraphicsManag...
InitializeMainM...
InitializeQuick...
Update
Camera3D
Class
Felder
pAspectRatio
pFrustum
pLookAt
pPosition
pProjectionMat...
pViewMatrix
Eigenschaften
Frustum
LookAt
Position
ProjectionMatrix
ViewMatrix
Methoden
applyToPlane
Camera3D
MoveComplete
MoveLookAtTa...
MoveOrbital
SetLookAt
SetPos
update

`namespace DKE_Kinect.Entwicklung.Grafik`
Der Namespace repräsentiert die Organisation und Darstellung von grafischen Objekten und Modellen beziehungsweise von Objekten, die sich um die Logik von grafischen Objekten selbst kümmern.

`abstrakt class DKE_Kinect.Entwicklung.Grafik.ControlManaging.TransitionEffect<T>`
Die Klasse ist abstrakt und stellt Eigenschaften und Methoden zur Verfügung, die für Effekte über eine bestimmte Zeitdauer andauernd elementar sind.
Die abstrakte Methode
```
protected abstract void UpdateValues (TransitionInterpolation InPol)
```
muss in erbenden Klassen implementiert werden. Sie soll alle Zwischenwerte berechnen.
Für die Zwischenwertberechnungen können verschiedene Interpolationen implementiert werden. Einige sind mit der Enumeration `enum TransitionInterpolation` vorgegeben.

`class DKE_Kinect.Entwicklung.Grafik.ControlManaging.TextureAlphaFadeTransition`
Die Klasse erbt von `TransitionEffect<float>` und beschreibt eine Übergangseffekt von einer Textur zu einer anderen, indem der Alpha – Wert der einen Textur mit der Zeit abnimmt.

`class DKE_Kinect.Entwicklung.Grafik.ControlManaging.Cursor3D`
Die Klasse beschreibt den Cursor im 3 - dimensionalen Raum. Er ist durch eine Kugel auf dem XNA – Screen sichtbar und kann auf eine bestimmte Position gesetzt werden oder um einen bestimmten Vektor verschoben werden.

`class DKE_Kinect.Entwicklung.Grafik.ControlManaging.Camera`
Repräsentiert die Kamera der 3 – dimensionalen Umgebung. Zu den Kameraeigenschaften gehören Position, Blickrichtung sowie die Matrizen für die Blickrichtung und die Projektion.

`class DKE_Kinect.Entwicklung.Grafik.ControlManaging.GraphicsManager`
Von dieser Klasse gehen alle grafischen Berechnungen aus, die auf dem XNA – Screen zu sehen sein werden sowie die dafür notwendigen logischen Objekte, wie Kamera, Cursor, Hauptmenü, Farbmenü etc..
Die Zeichenmethoden wurden in Spritebatch - Methoden und 3D – Zeichen - Methoden unterteilt, um sie logisch voneinander zu trennen.

Erweiterungsmöglichkeiten

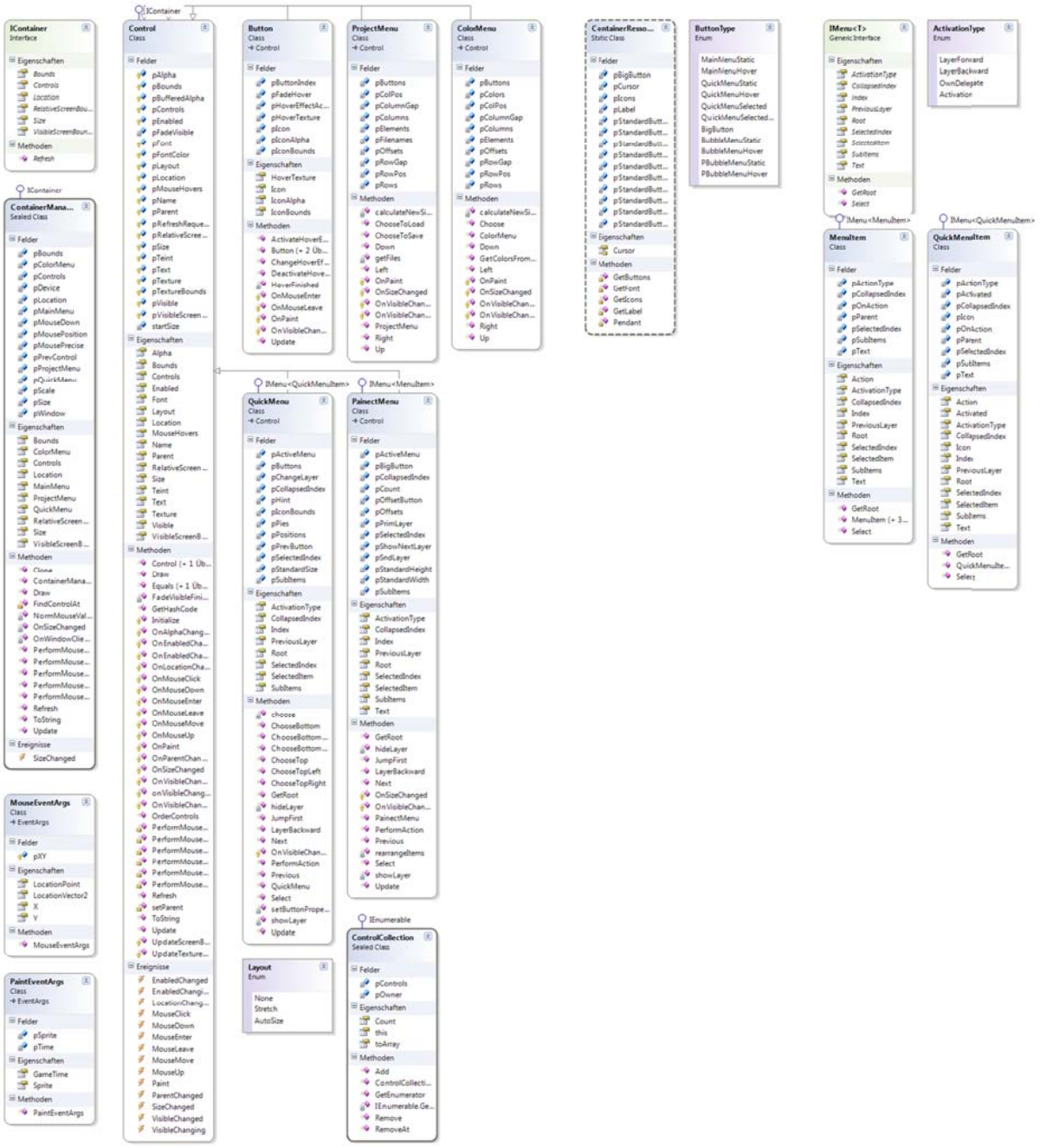
Weitere Effekte

Ein weiterer Effekt kann beispielsweise ein Bewegungseffekt sein.

Dazu muss eine neue Klasse angelegt werden, die im Groben so aussehen könnte:

```
class MoveEffekt : TransitionEffekt<Point>
{
	protected override void UpdateValues (TransitionInterpolation InPol)
	{
		switch (InPol)
		{
			case TransitionInterpolation.Linear:
				this.pMerged = new Point(
				  MathHelper.Lerp(
				    base.pOrig.X, base.pDest.X, this.Percent),
				  MathHelper.Lerp(
				    base.pOrig.Y, base.pDest.Y, this.Percent));
			  break;
			// more cases
			}
		}
	//...
}
```

ControlManaging



`namespace DKE_Kinect.Entwicklung.Grafik.ControlManaging`
Der Namespace repräsentiert eine geschlossene Einheit zur Anordnung von grafischen Steuerelementen. Hauptelemente sind einerseits die `Control` – Klasse, die in der Vererbungshierarchie Basis für alle anzuzeigenden Steuerelemente ist und andererseits die `ContainerManager` – Klasse, die alle anzuzeigenden Elemente enthält, weswegen beide aus dem `IContainer` – Interface erben.
Weiterhin bietet der Namespace mit der relativ umfangreichen Klasse `Control` viele individuelle Anpassungsmöglichkeiten. Im Folgenden sollen die wichtigsten Klassendetails kurz beschrieben und erläutert werden.

`class DKE_Kinect.Entwicklung.Grafik.ControlManaging.IContainer`
Stellt alle Basiseigenschaften für die Anzeige von Steuerelementen zur Verfügung.

`class DKE_Kinect.Entwicklung.Grafik.ControlManaging.Control`
Erbt von dem `IContainer` – Interface. Stellt eine Grundlage an Methoden, Eigenschaften und Events zur Verfügung, mit denen sich Texte und Texturen anzeigen lassen.
In der Klasse existieren die
```
public virtual void Update (GameTime gameTime)
```
und
```
public virtual void Draw (GameTime gameTime, SpriteBatch Sprite)
```
- Methode, die das Steuerelement aktualisieren und auf dem jeweiligen Parent zeichen.
Ein Steuerelement kann `Text` und / oder eine `Texture` enthalten, die beim Zeichnen angezeigt werden. Die Sichtbarkeit kann durch die Eigenschaft `Alpha` modifiziert werden.
Um Steuerelemente zu einem Steuerelements selbst hinzuzufügen, kann für ein entsprechendes Steuerelement `Control ctrl` folgende Funktion aufgerufen werden:
```
ctrl.Controls.Add(ctrl2);
```
Um die Konsistenz zwischen den Steuerelementen und den über- bzw. untergeordneten Elementen zu gewährleisten, wurde die Klasse `ControlCollection` extra für diese Klasse selbst implementiert.

`class DKE_Kinect.Entwicklung.Grafik.ControlManaging.ContainerManager`
Erbt von dem `IContainer` – Interface. Die Klasse verwaltet die Steuerelemente auf dem XNA – Screen, gerade Projektspezifische Steuerelemente wie beispielsweise das Hauptmenü. Auch diese Klasse besitzt die
```
public virtual void Update (GameTime gameTime)
```
und
```
public virtual void Draw (GameTime gameTime, SpriteBatch Sprite)
```
- Methode.
Der ContainerManager aktualisiert und zeichnet alle untergeordneten Steuerelemente mit genau diesen Methoden, indem er auch diese Methoden der Steuerelemente aufruft, sofern die Eigenschaft `Visible auf true gesetzt ist.`

`class DKE_Kinect.Entwicklung.Grafik.ControlManaging.Button`
Erbt von der `Control` – Klasse. Die Klasse stellt eine Schaltfläche als Steuerelement dar. Damit unter anderem die grafischen Merkmale wie das „TextureFading“ beim fokussieren der Schaltfläche sichtbar werden, wurde die überschreibbare Methode
```
protected override void OnPaint (PaintEventArgs e)
```
überschrieben. Darin werden alle Texturbezogen Zeichenvorgänge modifiziert oder neu geschrieben.

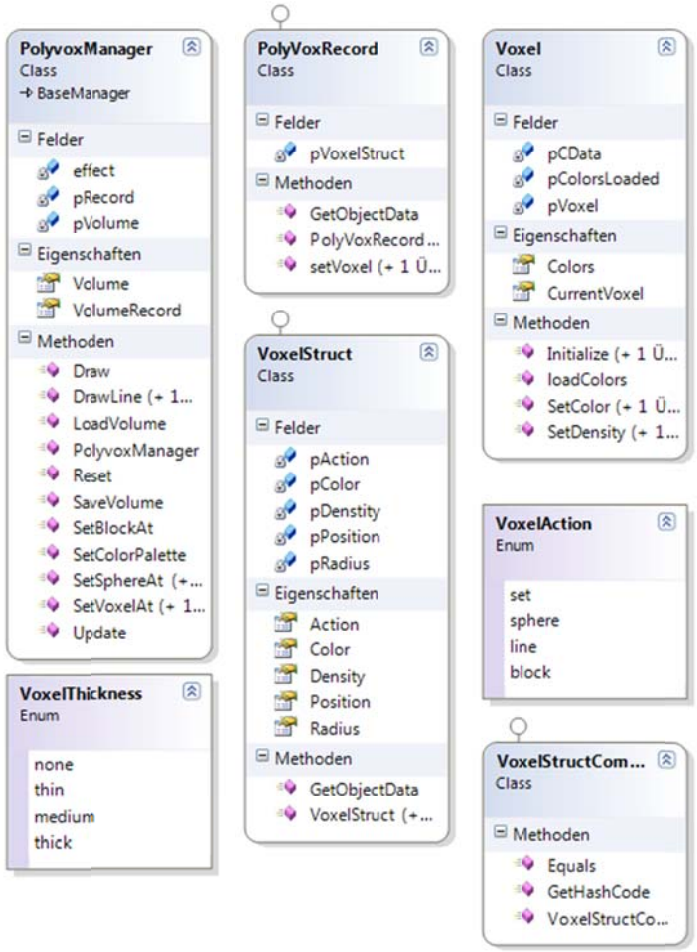
`class DKE_Kinect.Entwicklung.Grafik.ControlManaging.QuickMenu`
Erbt von der `Control` – Klasse und dem `IMenu<QuickMenuItem>` - Interface .
Das generische Interface enthält alle Menübezogenen Definitionen für die entsprechende Klasse, hier für `QuickMenuItem`. Zusätzlich zum Standardinterface enthält ein `QuickMenuItem` zusätzlich ein Icon, das abgespeichert wird, dass vom `QuickMenu` selbst angezeigt wird.
Andere Menüs sind ähnlich aufgebaut, weswegen für die Dokumentation nur diese Klasse in Betracht gezogen werden soll. Für einen Vergleich bitte in den Sourcecode schauen (`PainectMenu`, `ProjectMenu`, `ColorMenu`).
Das Menü besitzt die Standardeigenschaften wie `SelectedItem`, `SelectedIndex`, `Root` etc.
Da die Eigenschaft `SubItems`, mit der auf die einzelnen Menüpunkte zugegriffen wird, vom Typ `List<QuickMenuItem>` ist, können wie bei jeder `List<T>` mit den Methoden `Add`, `Insert`, `Remove` etc. die Elemente erweitertet und bearbeitet werden.
Aufgrund der festgelegten Anzahl an Elementen pro Layer kann jedes einzelnes Element mit einer jeweiligen Methode abgegriffen werden (z.B. `ChooseBottom()`).

Erweiterungsmöglichkeiten

Lokalisation im Programm

Bisher sind alle Texte für die Menüs in einer Sprache, deutsch, hardgecodet implementiert.
Um mehrere Sprachen in den Menüs anzeigen zu lassen, besteht die Möglichkeit, ein jeweiliges Sprachpaket in einer XML – Datei abzuspeichern, beispielweise für jedes Menü einen einzelnen Tag.
Bei Programmstart muss dann vorerst die jeweilige Sprache geladen werden und die einzelnen `Text` – Eigenschaften der Menüpunkte gesetzt werden.
Hilfsmittel dafür ist die Klasse `XMLDocument` aus dem Namespace `System.Xml`.

PolyVox


PolyvoxManager
Class
BaseManager
Felder
effect
pRecord
pVolume
Eigenschaften
Vclume
VclumeRecord
Methoden
Draw
DrawLine (+ 1...
LoadVolume
PclyvoxManager
Reset
SaveVolume
SetBlockAt
SetColorPalette
SetSphereAt (+...
SetVoxelAt (+ 1...
Update
VoxelThickness
Enum
none
thin
medium
thick
PolyVoxRecord
Class
Felder
pVoxelStruct
Methoden
GetObjectData
PolyVoxRecord...
setVoxel (+ 1 Ü...
VoxelStruct
Class
Felder
pAction
pColor
pDenstity
pPosition
pRadius
Eigenschaften
Action
Color
Density
Position
Radius
Methoden
GetObjectData
VoxelStruct (+...
Voxel
Class
Felder
pCData
pColorsLoaded
pVoxel
Eigenschaften
Colors
CurrentVoxel
Methoden
Initialize (+ 1 Ü...
loadColors
SetColor (+ 1 Ü...
SetDensity (+ 1...
VoxelAction
Enum
set
sphere
line
block
VoxelStructCom...
Class
Methoden
Equals
GetHashCode
VoxelStructCo...

Der `DKE_Kinect.Entwicklung.PolyVox` Namespace beinhaltet Klassen, welche für die Verwaltung und Modifikation der Volumina verantwortlich sind.
Herzstück ist der `PolyvoxManager`, welcher die Anbindung zur *PolyVox C++ Library* über einen *.Net-Wrapper* bietet. Er ist verantwortlich für das Setzen der verschiedenen Formen, initialisiert das Volumen und verwaltet das Extrahieren und Rendern des Modells.

Die Klasse `PolyVoxRecord` speichert eine Spiegelung des aktuellen Volumens in vereinfachter Form. Immer wenn ein Voxel durch den `PolyvoxManager` gesetzt wird, setzt dieser auch die zugehörige Struktur im `PolyVoxRecord`. Beim Speichern und Laden wird nur die vereinfachte Struktur abgelegt, aus welcher der `PolyVoxRecord` wieder ein Volumen im `PolyvoxManager` rekonstruieren kann.

Diese vereinfachte Struktur wird kodiert durch die Klasse `VoxelStruct`, in der die Position, die Dichte, das Material und die Aktion, also die Art des Voxels gespeichert wird.

Der `enum VoxelAction` listet die möglichen Aktionen eines Voxels: „*set*“ als einzelner Punkt, „*sphere*“ mit einem Radius als Kugel um einen Punkt, „*line*“ als Endpunkt einer Linie und „*block*“ als Eckpunkt eines Quaders.

Die `class Voxel` beinhaltet den aktuelle verwendeten „Pinsel“, hier sind Dichte und Material gespeichert, mit denen neu gesetzte Voxel gefüllt werden. Außerdem kann über die `Voxel` Klasse Einfluss auf die Farbpalette genommen werden.

Erweiterungsmöglichkeiten

Neue Formen definieren, z.B. Curve

Benötigt Umwandlung von einer Curve in bekannte Form, z.B. Approximation durch Linien.

Schritte:

- Klasse die eine Kurve speichern kann (Bezier-Kurve)
- Rasterung dieser Kurve in n Linien
- Setzen der Linienteile im `PolyvoxManager` mit:

```
public void DrawLine (Vector3 from, Vector3 to)
```

Neue Farben

In Painect stehen immer 64 versch. Farben zur Verfügung. Um eigene Farben zu laden, wird eine Palette der Größe 64x1 Pixel benötigt. Diese Palette wird in die `Voxel` Klasse geladen:

```
public static void loadColors(Texture2D colorP)
```

Über die Funktion `public static void SetColor(Color c)` lässt sich jede Farbe approximieren, die `Voxel` Klasse wählt dann automatisch die Farbe der Palette aus, welche am ehesten *C* entspricht.

Neue Funktion UNDO

Um UNDO zu ermöglichen, müsste eine neue `VoxelAction` in der Klasse `VoxelStruct` eingeführt werden, welche ein UNDO codiert.
Der `PolyVoxRecord` müsste insofern erweitert werden, dass er beim Auftreten der UNDO Aktion diese nicht einreiht, sondern je nach letzter Aktion 1 oder 2 Voxel entfernt (1 bei „SET“ und „SPHERE“, 2 bei „LINE“ oder „BLOCK“).
Als letztes muss im `PolyvoxManager` ein Funktion erstellt werden, welche einen neuen Voxel mit der UNDO Aktion setzt :

```
public void UNDO ()
{
	pRecord.setVoxel(Vector3.ZERO, Voxel.CurrentVoxel,VoxelAction.undo);
}
```