



VEKTORIA-MANUAL

FRAMES &

VIEWPORTS



Frames & Viewports

Inhalt



/// **UEBERSICHT**

/// **FRAMES**

/// **VIEWPORTS**

/// **SCHALTMETHODEN**

/// **STILE**

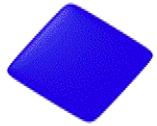


Kapitel 1

Übersicht über Frames und Viewports



UEBERSICHT



2 // / /

3 // / /

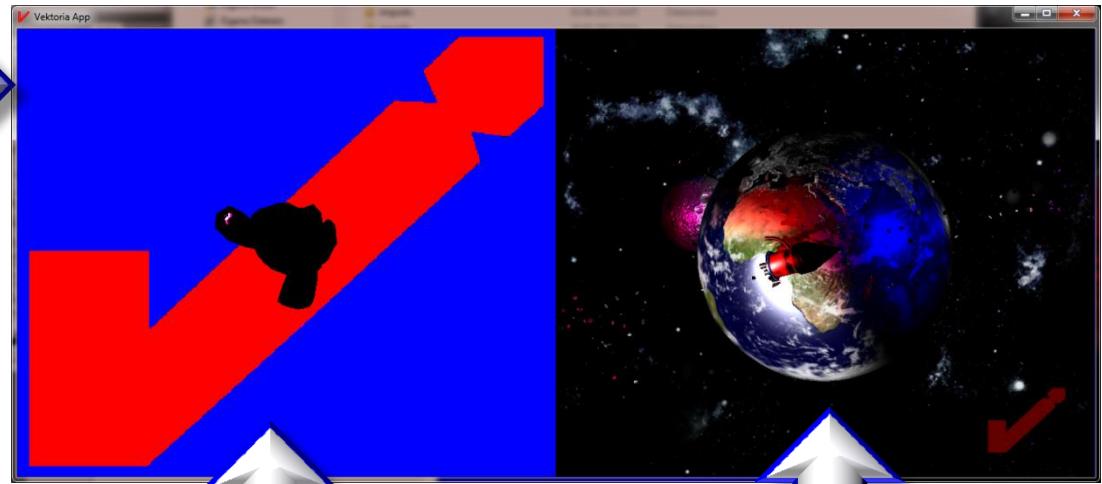
4 // / /

5 // / /



Frames & Viewports - Beispiel

 Frame



 Viewport 1

 Viewport 2



Kapitel 2

Kapitel 2



/// **FRAMES**





Was ist ein Frame?

- Ein Frame ist ein rechteckiger Renderbereich, der in einem Fensterrahmen (Window) läuft.
- Ein Frame hat eine Render-API (z.B. Direct3D, OpenGL oder NullRenderer) und eine Input-API (z.B. DirectInput, STL oder Null_Device).
- Auf einem Computer können bis zu vier Frames laufen (mehr macht sowieso keinen Sinn).
- Jeder Frame kann beliebig viele Viewports (Sichtfenster) und Devices (Ein- und Ausgabegeräte wie Maus, Tastatur, Game Controller) enthalten.

1 // / /

2 // / /

3 // / /

4 // / /

5 // / /



CFrame::Init

```
CRenderApi* Init(HWND hwnd,
    EApiRender eApiRender
        = eApiRender_DirectX11_shadermode150,
    EApiInput eApiInput
        = eApiInput_DirectInput,
    EApiSound eApiSound
        = eApiSound_DirectSound,
    EShaderCreation eShaderCreation
        = eShaderCreation_CompileChanges,
    EShaderAutoRecompilation eShaderAutoRecompilation
        = eShaderAutoRecompilation_Enabled);
```

Initialisiert den Frame und gibt einen Zeiger auf die automatisch kreierte RenderApi zurück. Der Methode muss als ersten Parameter zwingend ein ein Handle auf das Windowsfenster, in das CFrame hineinmalt übergeben werden (**hwnd**). Dazu existieren fünf optionale Parameter, die auf den nächsten Seiten detailliert erklärt werden:
eApiRender, **eApiInput**, **eApiSound**,
eShaderCreation und **eShaderAutoRecompilation**.

CFrame::Init (eApiRender)



eApiRender bestimmt die gewünschte Grafik-Basis-Api.

Zurzeit sind folgende Eingabewerte möglich:

- **eApiRender_DirectX11_Shadermodel41**
abgespeckter DirectX11-Renderer mit Shader Modell 4.1,
diese Option ist sinnvoll, wenn Sie einen älteren Rechner
(um 2009) besitzen, der nicht Shadermodell 5.0 unterstützt.
- **eApiRender_DirectX11_Shadermodel50**
DirectX11-Renderer mit Shader Modell 5.0,
dies ist die Default-Einstellung
- **eApiRender_DirectX11_ForwardPlus**
bester Renderer mit ForwardPlus-Technologie,
basiert ebenfalls auf DirectX11 mit dem Shader Modell 5.0
allerdings ist die Parameterjustierung bei Lichtern komplex.
- **eApiRender_Null**
Null-Renderer, bei dem die Renderanweisungen ins Leere
laufen, ist nur sinnvoll zum Debuggen.

CFrame::Init (eApiSound)

eApiSound bestimmt die gewünschte Grafik-Basis-Api.

Zurzeit sind folgende Eingabewerte möglich:

- **eApiSound_DirectSound**
Es wird für Musik und Klänge und Geräusche die DirectSound-Api geladen (Default)
- **eApiSound_DirectAudio**
Es wird für Musik und Klänge und Geräusche die DirectAudio-Api geladen
- **eApiSound_OpenAL**
Es wird für Musik und Klänge und Geräusche die OpenAL-Api geladen
- **eApiSound_Null**
Null-Sound-Api, bei dem die Soundanweisungen ins Leere laufen, ist nur sinnvoll zum Debuggen.



CFrame::Init-Parameter

eApilInput gibt die gewünschte Eingabe-API an.

Zurzeit sind zurzeit folgende Optionen erlaubt:

- **eApilInput_DirectInput (Default)**
Die Eingabe-API basiert auf DirectInput.
- **eApilInput_Null**
Die Eingabe-Abfragen laufen hier ins Leere.
Nur Sinnvoll zum Debuggen.



CFrame::Init-Parameter

eShaderCreation gibt die Art der Shadererzeugung an.

Es sind drei Optionen möglich:

- **eShaderCreation_ForceCompile**

Hier werden alle Shader zwingend am Anfang neu kompiliert. Damit startet das Programm aber wesentlich später.

- **eShaderCreation_CompilerChanges**

Hier werden nur diejenigen Shader kompiliert, die sich seit dem letzten Mal geändert haben. Dies ist die Default-Option

- **eShaderCreation_UseCached**

Hier wird kein Shader übersetzt, sondern nur die schon übersetzten Shader aus dem Cache verwendet. Veränderungen am Shader-Source-Code wirken sich daher nicht auf die Applikation aus.



CFrame::Init-Parameter

1 // / / / **eShaderAutoRecompilation** gibt den Zeitpunkt der Shader-Kompilierung an.

Wirkt sich naturgemäß nicht aus, falls der vorherige Parameter eShaderCreation auf eShaderCreation_UseCached gesetzt wurde.

2 // / / / Es sind folgende Optionen möglich:

- **eShaderAutoRecompilation_Disabled**
Es werden die Shader nur beim Programmstart kompiliert.

- **eShaderAutoRecompilation_Enabled**
Hier werden Shader automatisch rekompiliert, wenn sie verändert wurden, auch während der Laufzeit. Dies ist der Default-Wert.

3 // / / /

4 // / / /

5 // / / /



Kapitel 3

Kapitel 3



1 // / / /

2 // / / /

///VIEWPORTS



4 // / / /

5 // / / /





Viewport-Initialisierung

Will man einen Viewport erzeugen, der den ganzen Frame ausfüllt (Normalfall), dann verwendet man folgende Initialisierungsmethode:

void InitFull(CCamera * pcamera);



Hier wird ein Pointer auf die Kamera übergegeben, deren Sicht das Viewport anzeigen soll.



Viewport-Initialisierung



Manchmal soll ein Viewport nur einen Teil des Frames ausfüllen,
z.B. für:

- Split-Screens
- In-Screens
- Dreitafelprojektionen
- Triptychon-Darstellung
- etc.

Dafür verwendet man die Initialisierungsfunktion Init:

```
void Init(CCamera * pcamera,
          float frx, float fry, float frwidth,
          float frHeight);
```



Viewports-Initialisierung

Die Größe eines Viewports bezieht sich immer auf den übergeordneten Frame.

Die Größe wird bei der Initialisierung fraktional angegeben.
Dabei ist die linke obere Ecke gleich ($x=0.0, y=0.0$) und die rechte untere Ecke gleich ($x=1.0, y=1.0$)

Beispiel:

So wird ein Spilt-Screen erzeugt:

```
m_rvLeft.Init(&m_zcLeft, 0.0f, 0.0f, 0.5f, 1.0f);  
m_rvRight.Init(&m_zcRight, 0.5f, 0.0f, 0.5f, 1.0f);
```



Sehr praktisch: Wird ein Frame vergrößert oder verkleinert ändern sich automatisch alle seine Viewports sinnfällig mit.



Viewport-Ansichten



Für ein Viewport können verschiedene Rendering-Parameter ein- und ausgeschaltet werden.



Sie können auch in einem bestimmten Stil gerendert werden.



Kapitel 4

Kapitel 4

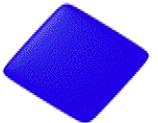


1 // / / /

2 // / / /

3 // / / /

/// V.-// SCHALTMETHODEN // / /



5 // / / /





Viewport-Schaltmethoden



1 // / / /
void SetBackfaceCullingOn();
void SetBackfaceCullingOff();

Schaltet das Backface-Culling für den Viewport an oder aus (Default = an)

2 // / / /
void SetAntialiasingOn();
void SetAntialiasingOff();

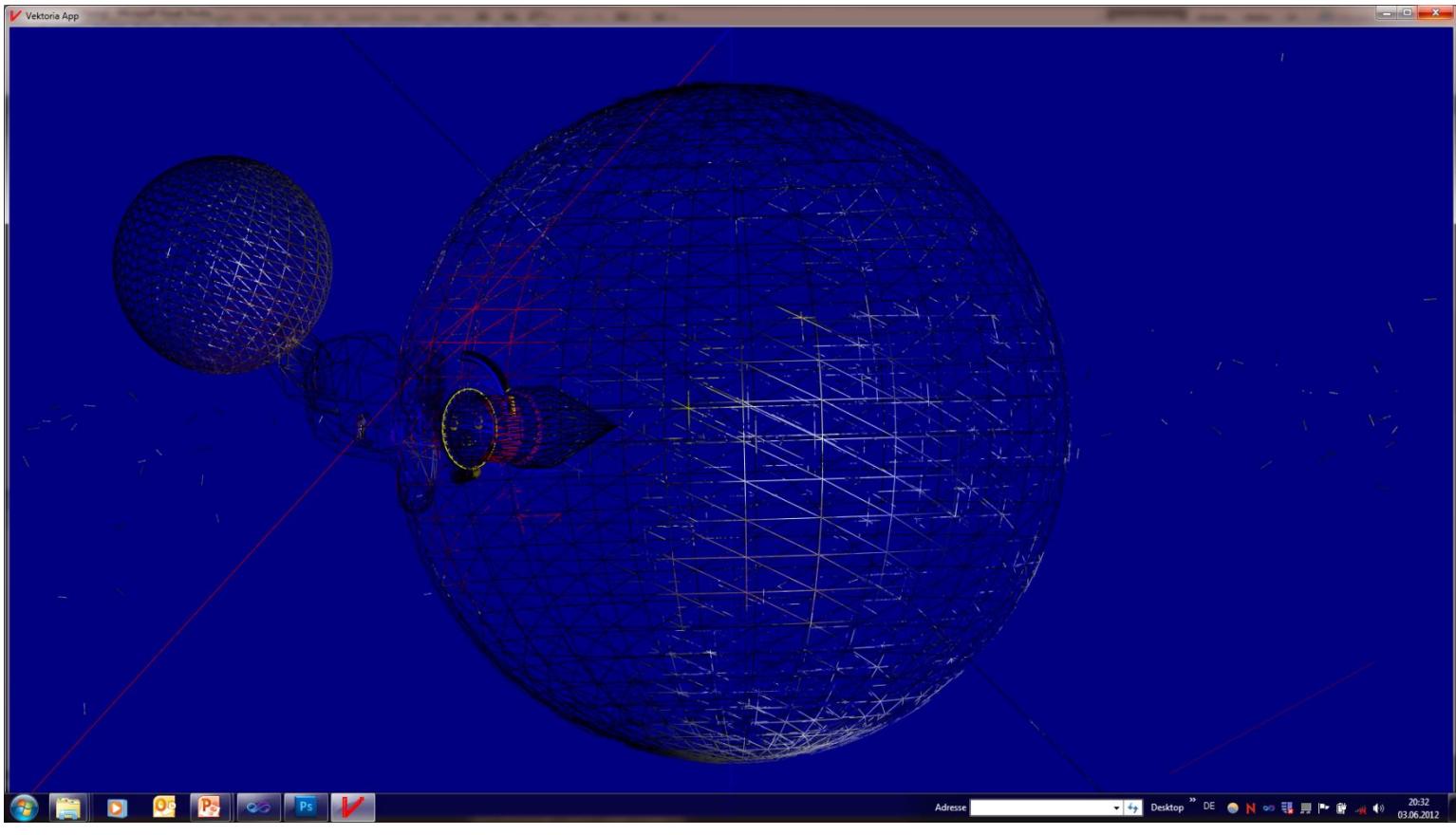
Schaltet Antialiasing für den Viewport an oder aus (Default = an), funktioniert natürlich nur, wenn die Grafikkarte dies unterstützt.

3 // / / /
void SetWireframeOn();
void SetWireframeOff();

Schaltet Drahtgittermodellansicht an oder aus (Default = aus).

Viewport-Schaltmethoden

SetWireframeOn

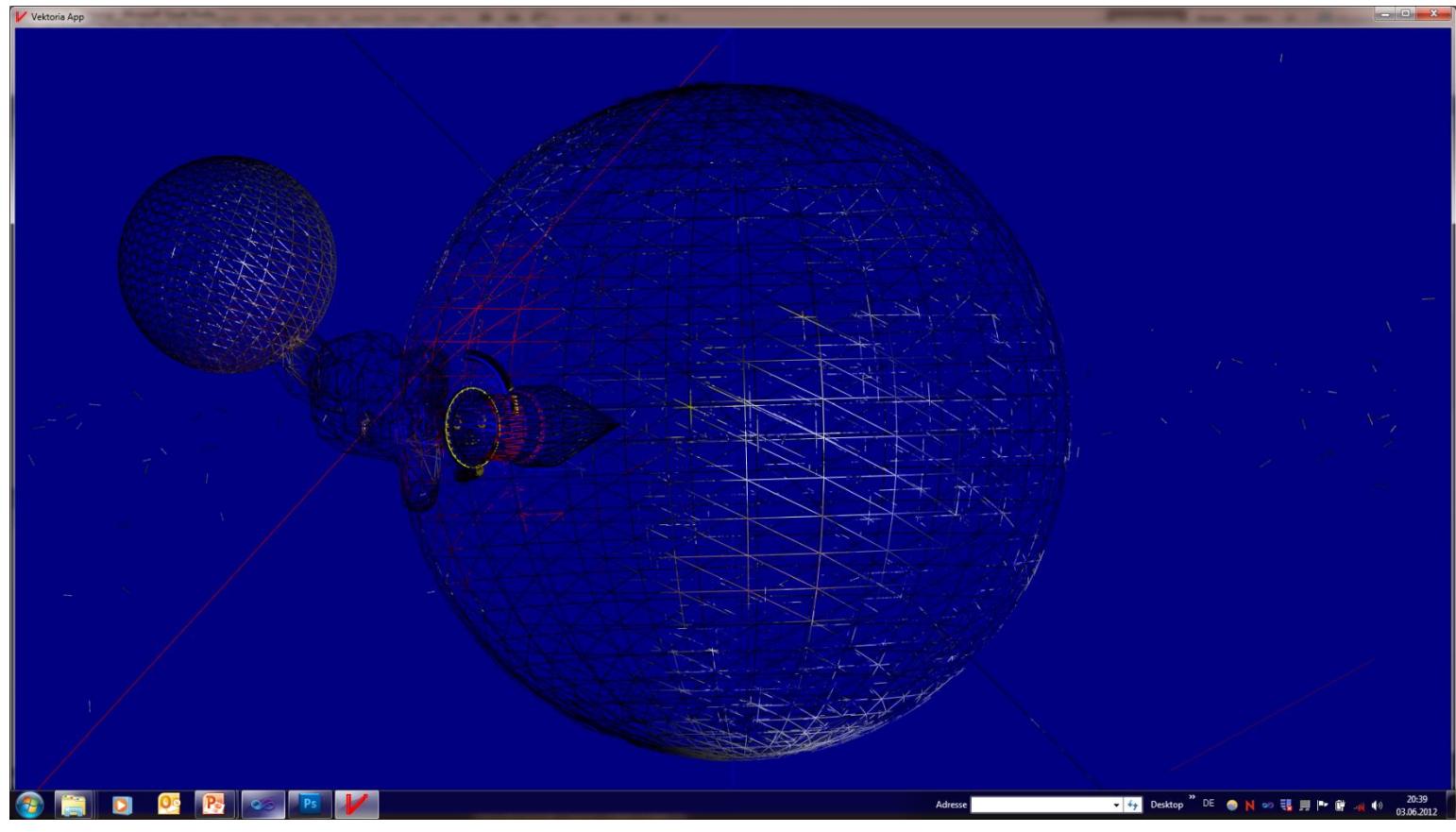


1 // / / /
2 // / / /
3 // / / /
4 // / / /
5 // / / /



Viewport-Schaltmethoden

WireframeOn+BackfaceCullingOff



1 // / / /

2 // / / /

3 // / / /

4 // / / /

5 // / / /



Weitere Viewport-Schaltmethoden



1 // / / /

2 // / / /

3 // / / /

4 // / / /

5 // / / /

```
void SetShadowRenderingOn();
// Schaltet Schatten für den Viewport an (Default: an)
void SetShadowRenderingOff();
// Schaltet Schatten für den Viewport aus (Default: an)
void SetPointLightRenderingOn();
// Schaltet Punktlichter für den Viewport an (Default: an)
void SetPointLightRenderingOff();
// Schaltet Punktlichter für den Viewport aus (Default: an)
void SetParallelLightRenderingOn();
// Schaltet Parallellichter für den Viewport an (Default: an)
void SetParallelLightRenderingOff();
// Schaltet Parallellichter für den Viewport aus (Default: an)
void SetSpotLightRenderingOn();
// Schaltet Schweiwerferlichter für den Viewport an (Default: an)
void SetSpotLightRenderingOff();
// Schaltet Scheinwerferlichter für den Viewport aus (Default: an)
```



Weitere Viewport-Schaltmethoden



1 // / / /
2 // / / /
3 // / / /
4 // / / /
5 // / / /

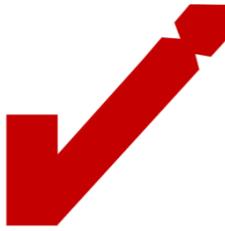
```
void SetShadowRenderingOn();
void SetShadowRenderingOff();
void SetPointLightRenderingOn();
void SetPointLightRenderingOff();
void SetParallelLightRenderingOn();
void SetParallelLightRenderingOff();
void SetSpotLightRenderingOn();
void SetSpotLightRenderingOff();
```

Weitere Schaltmethoden des Viewports, um Lichter und Schatten zu Anschauungszwecken aus- und anzuschalten!

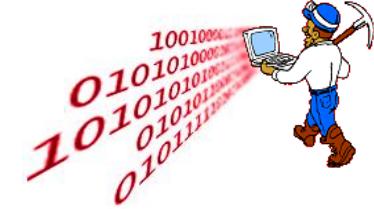


Achtung, da aus technischen Gründen die obigen Schaltmethoden die jeweiligen Features nur zu Anschauungszwecken unterdrücken, wird keine Framerate-Beschleunigung erzielt!
Diese Schaltmethoden sollten daher nur in Ausnahmefällen verwendet werden! Will man kein Licht oder Schatten haben, einfach nicht implementieren!





Übung zu Viewports



Erzeugen Sie einen Split-Screen, wobei der linke Viewport die gleiche Kameraansicht rendern soll, wie die rechte, nur als Drahtgittermodell!

Freiwillige Zusatzaufgabe für die schnellen Nerds:

- Lassen Sie die Teilungslinie des Split-Screens während der Laufzeit variieren!



Kapitel 5

Kapitel 5



1 // / / /

2 // / / /

3 // / / /

4 // / / /

/// V.-// STILE





Ein Viewport kann in einem bestimmten Stil gerendert werden. Die Betreffenden Methoden der Klasse CViewport fangen mit „Style“ an, z.B.:

1 // / / /
2 // / / /
3 // / / /
4 // / / /
5 // / / /

```
m_zv.StyleSepia(); // der Viewport m_zv  
// wird jetzt in Sepia  
// gezeichnet
```



Stile und Schaltmethoden können beliebig miteinander kombiniert werden.

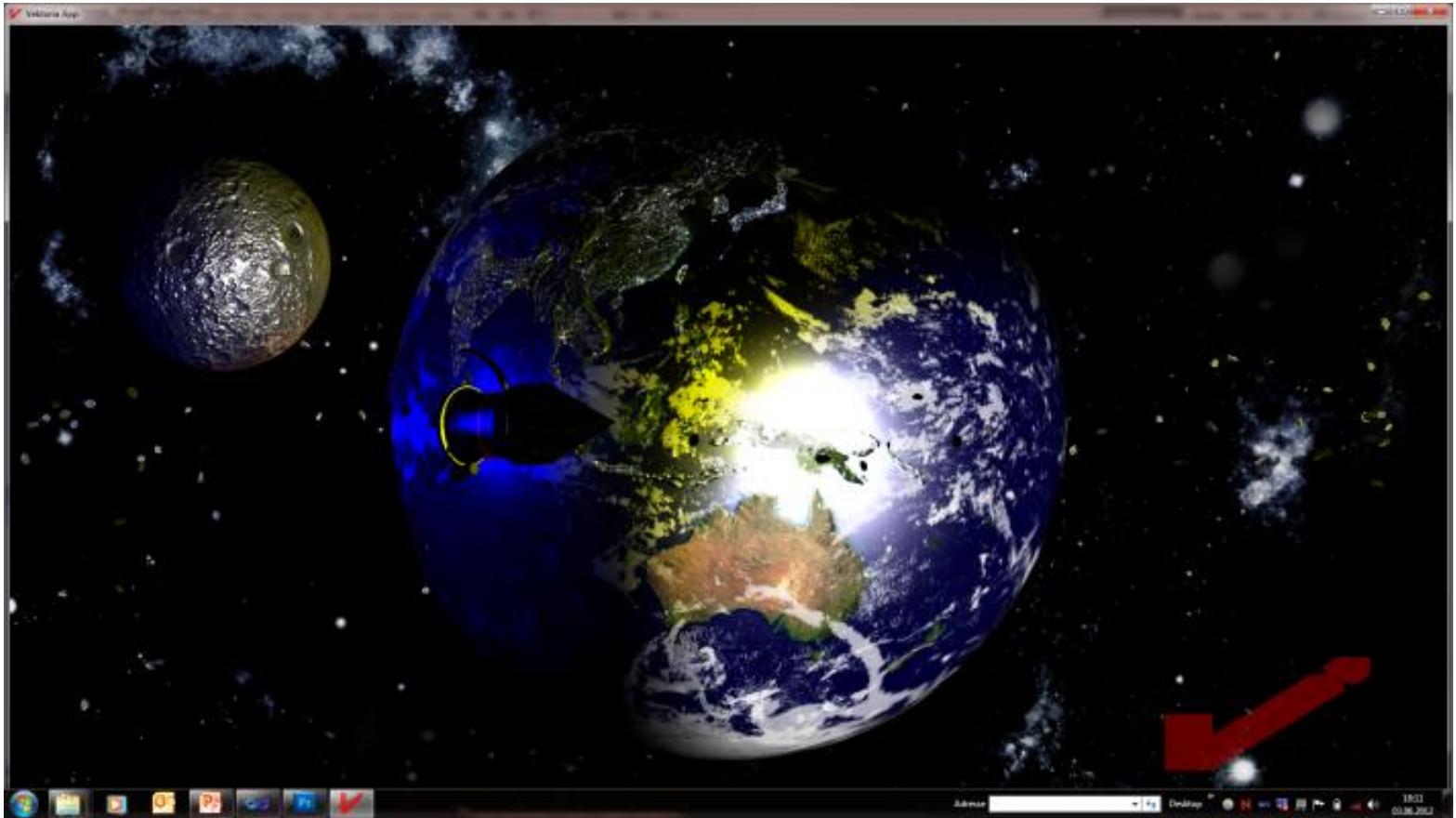


Stile können auch während der Laufzeit geändert werden.

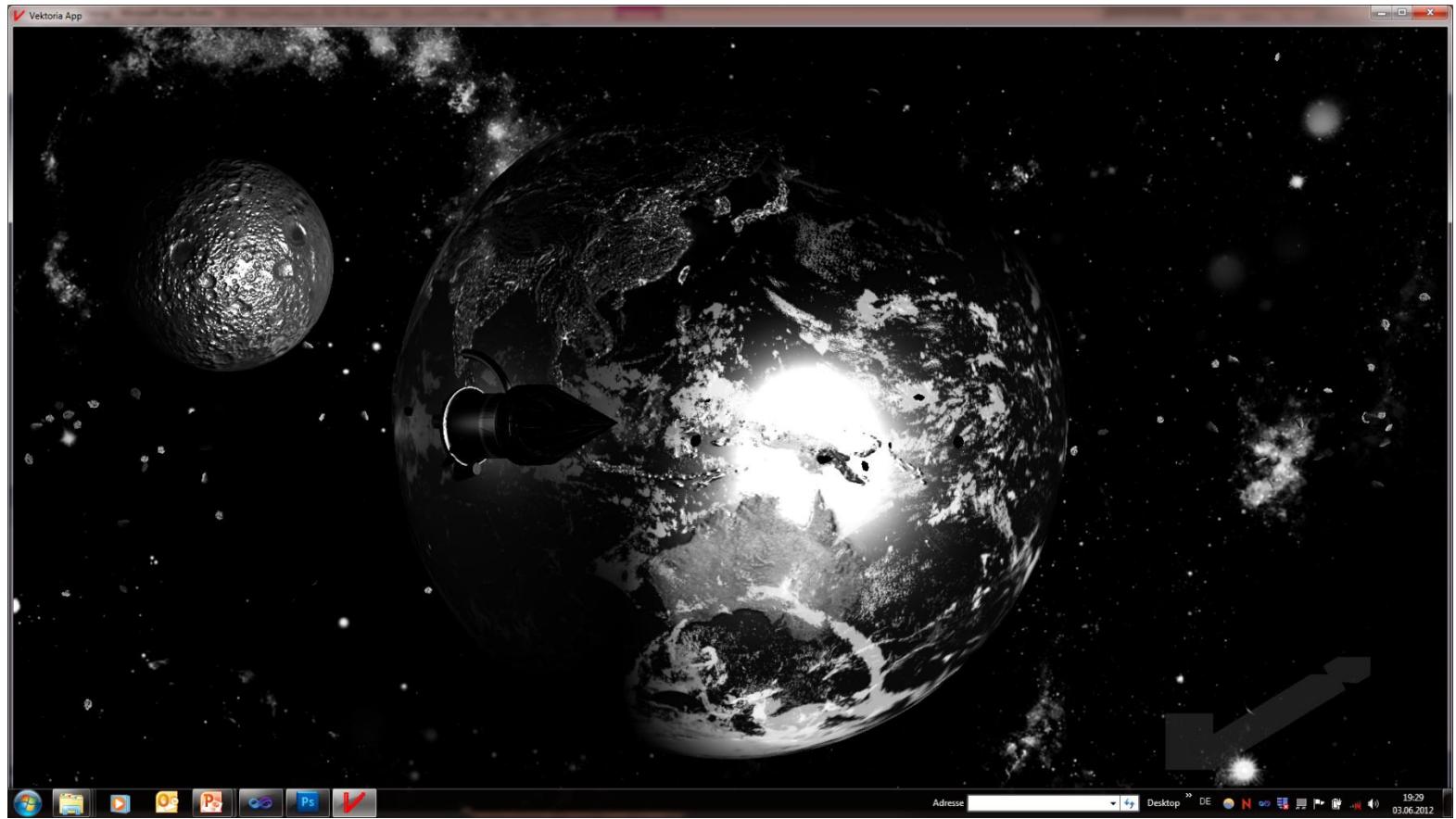
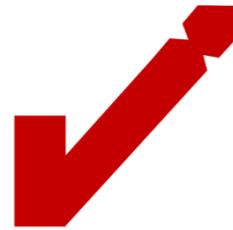


Viewportstile

Ohne Style

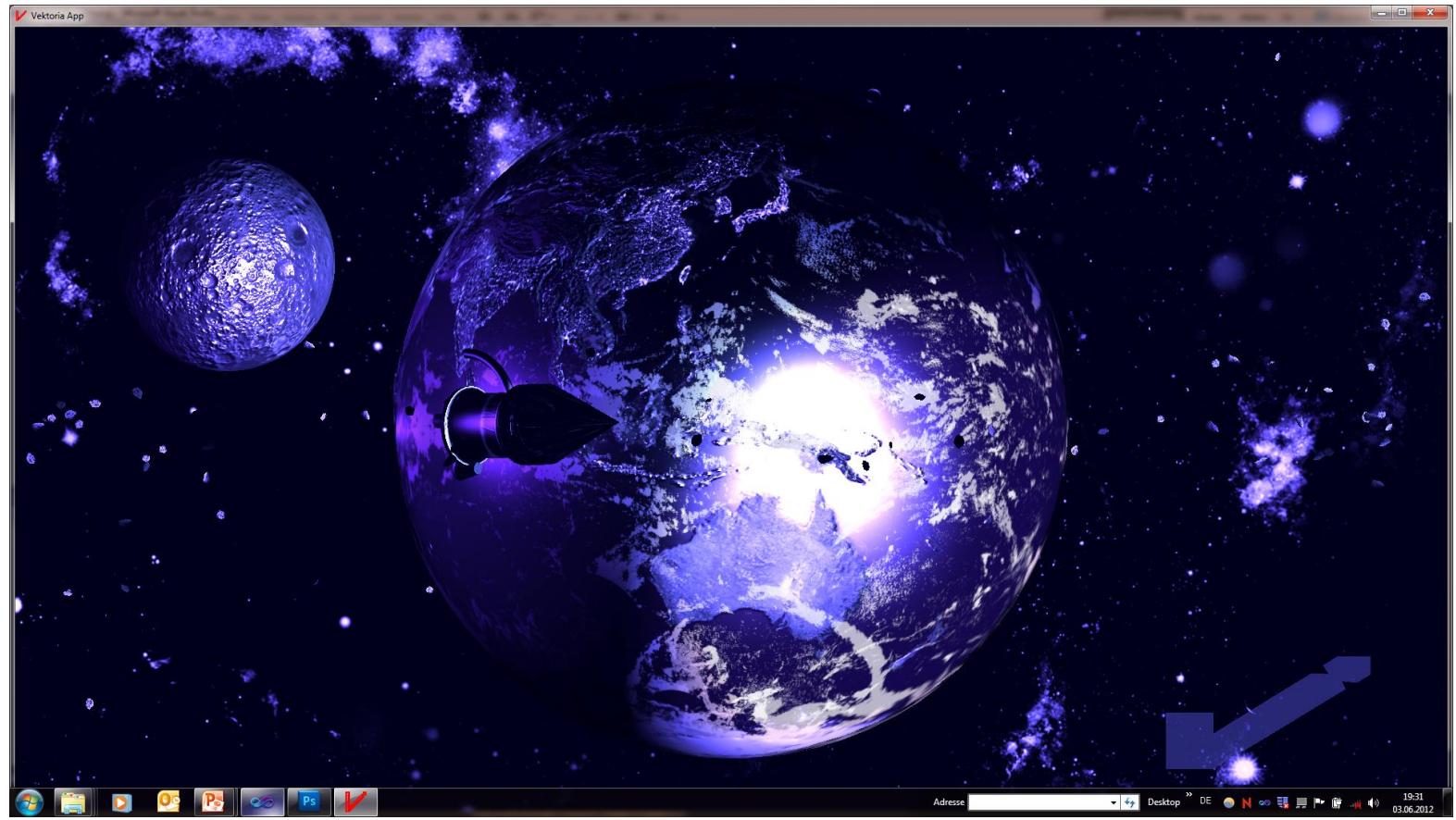


Viewportstile StyleBW



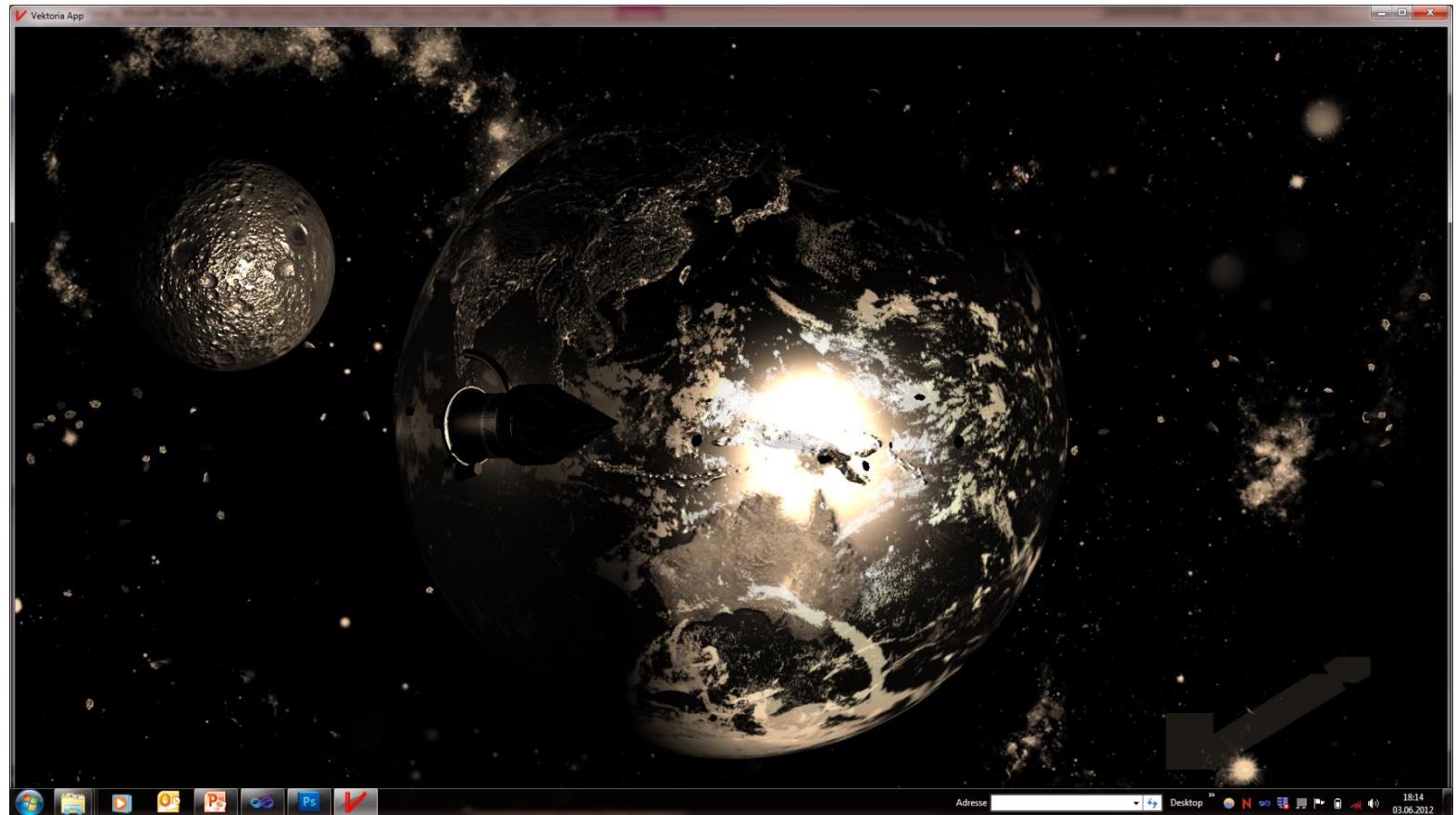
Viewportstile

StyleBlueDream



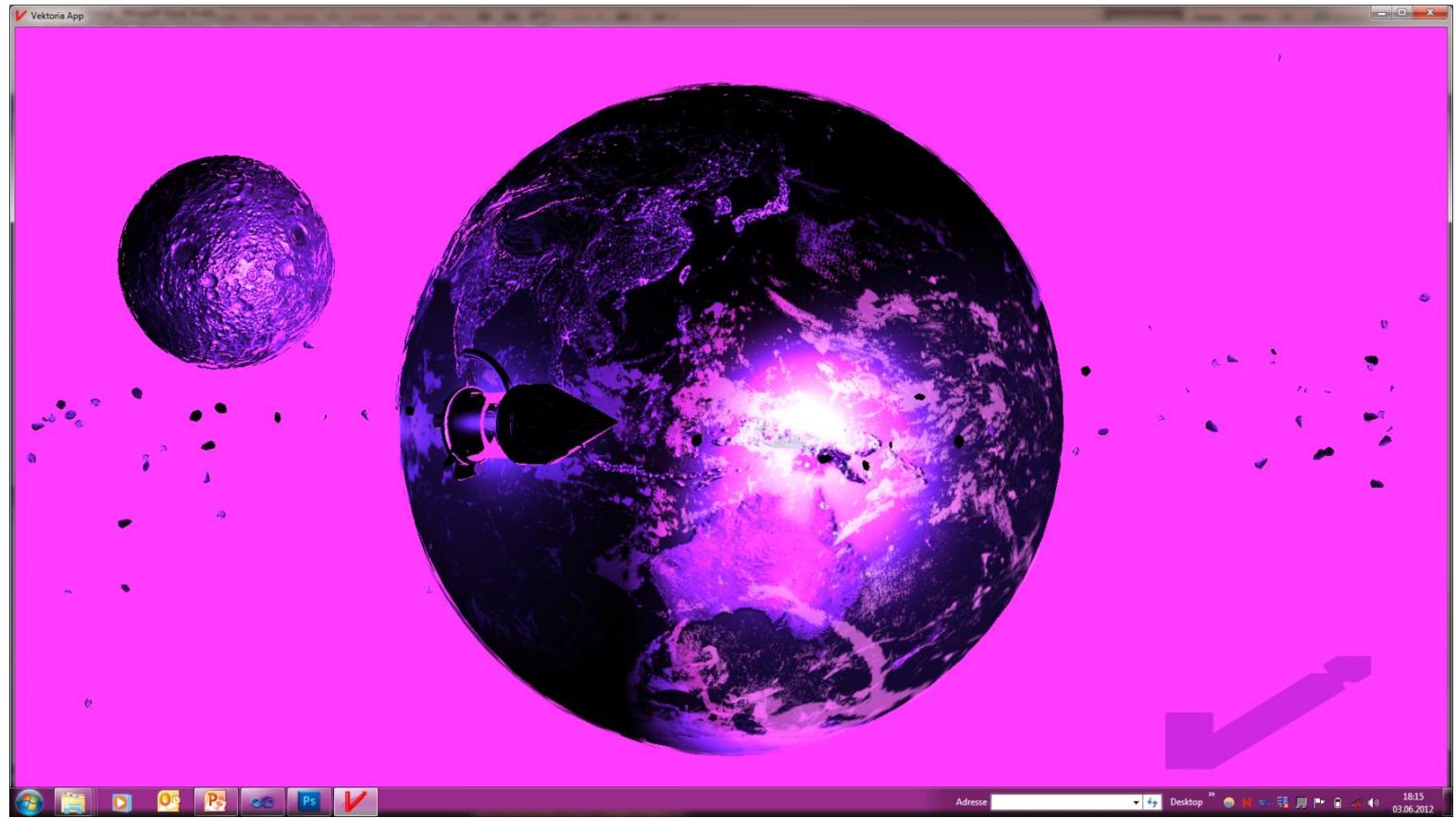
Viewportstile

StyleSepia



Viewportstile

StylePurpleHaze

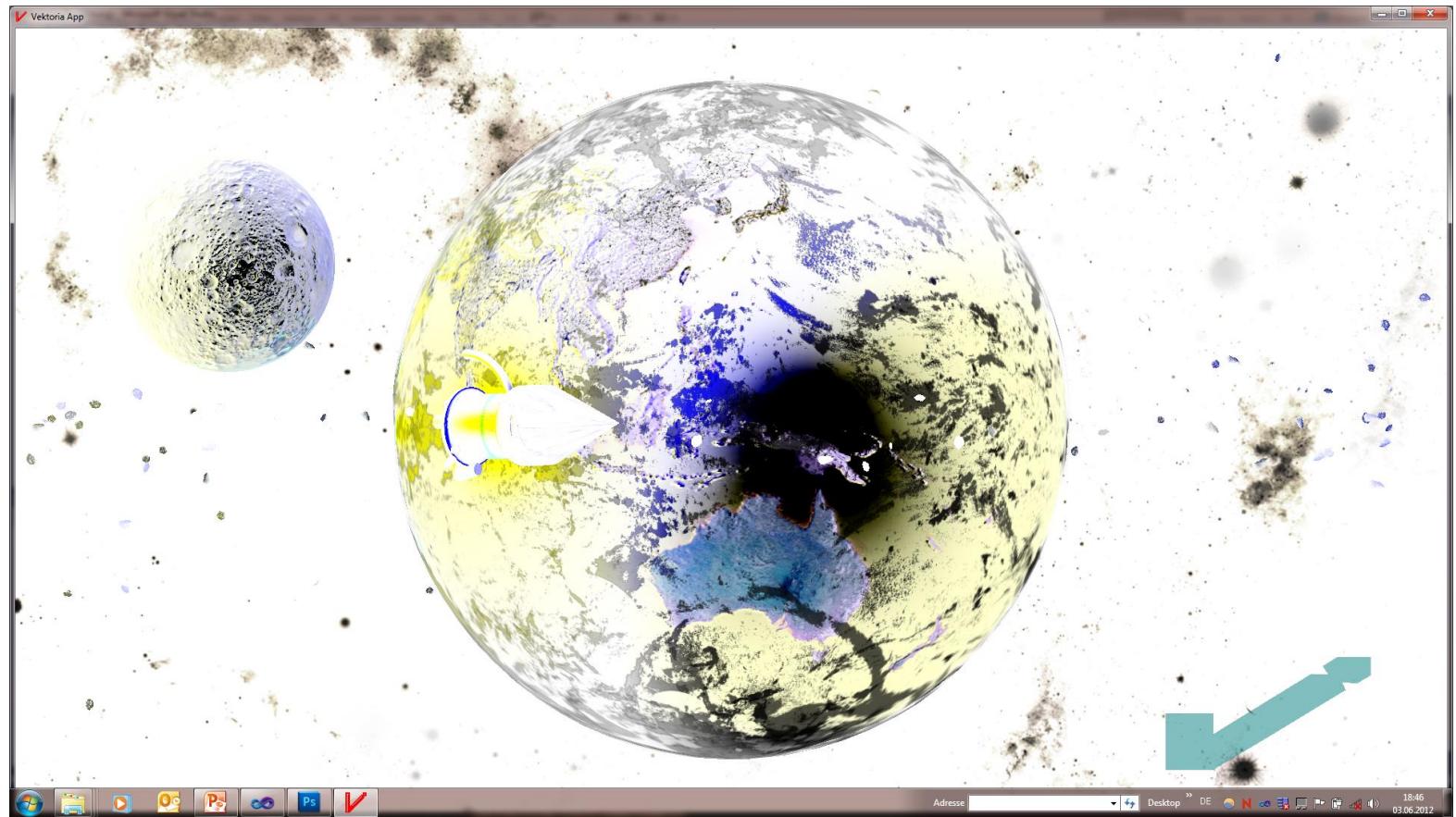


Viewportstile

StyleInverse

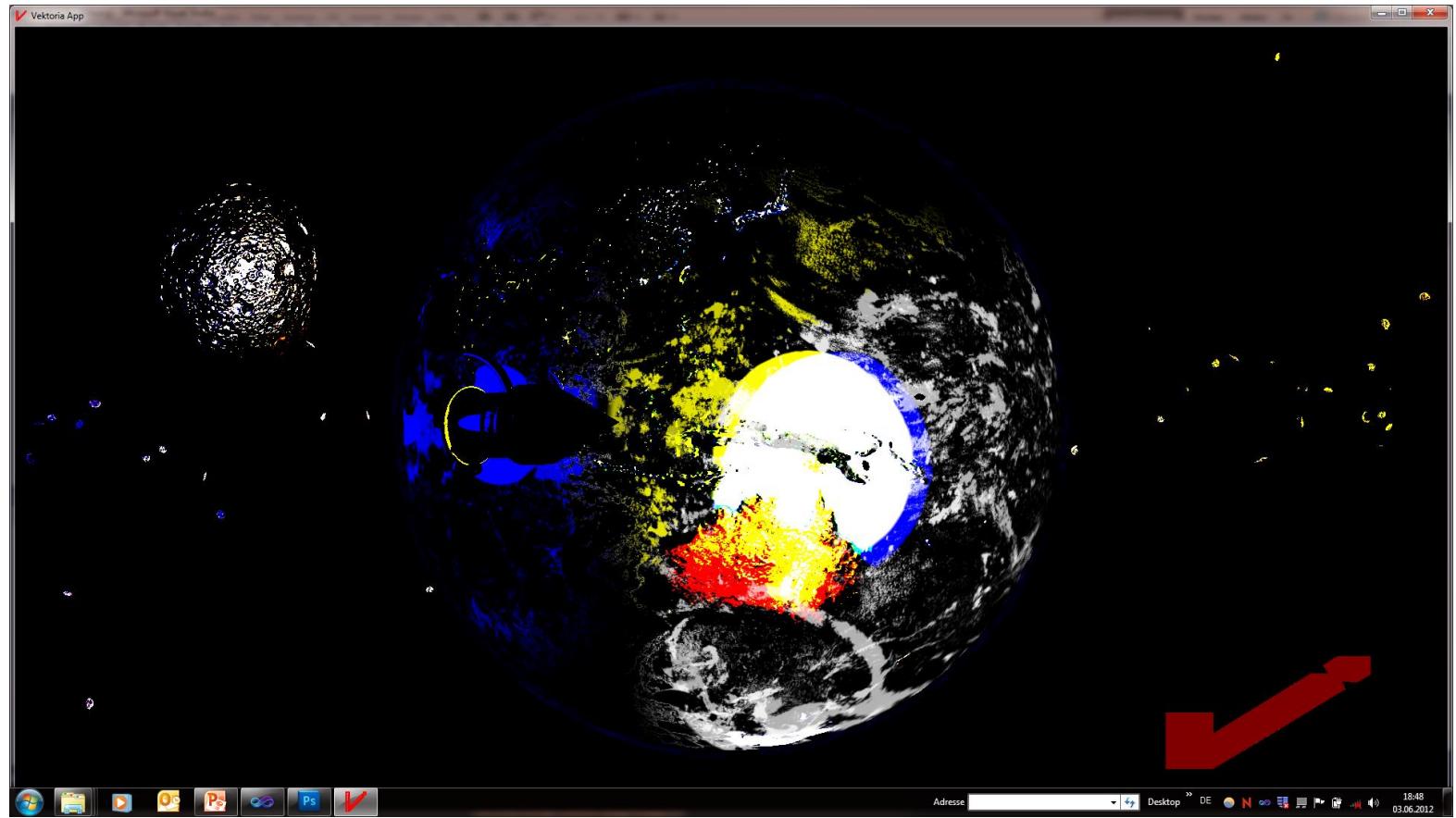


- 1 // / / /
- 2 // / / /
- 3 // / / /
- 4 // / / /
- 5 // / / /



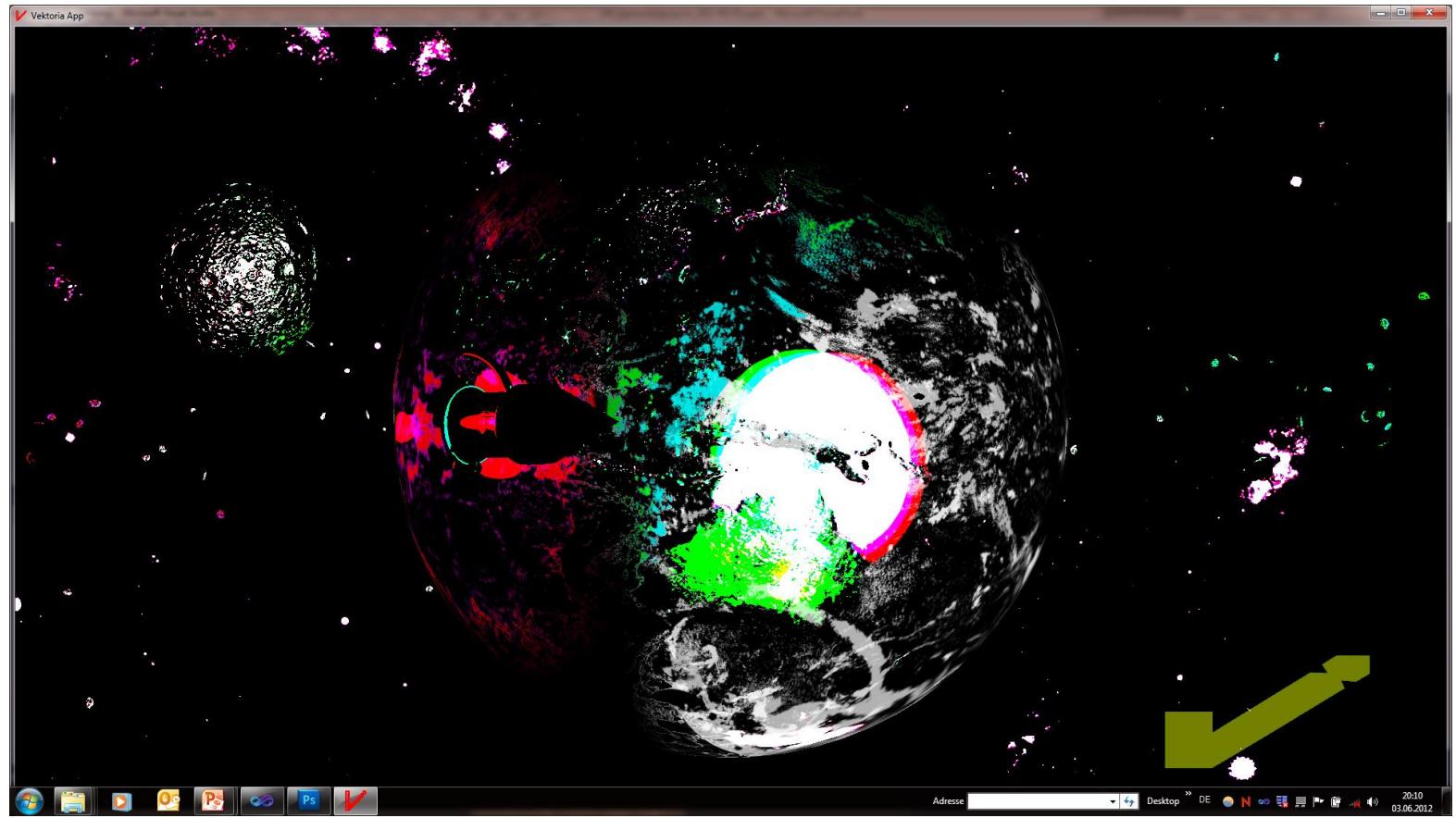
Viewportstile

StyleFloppyPoppy



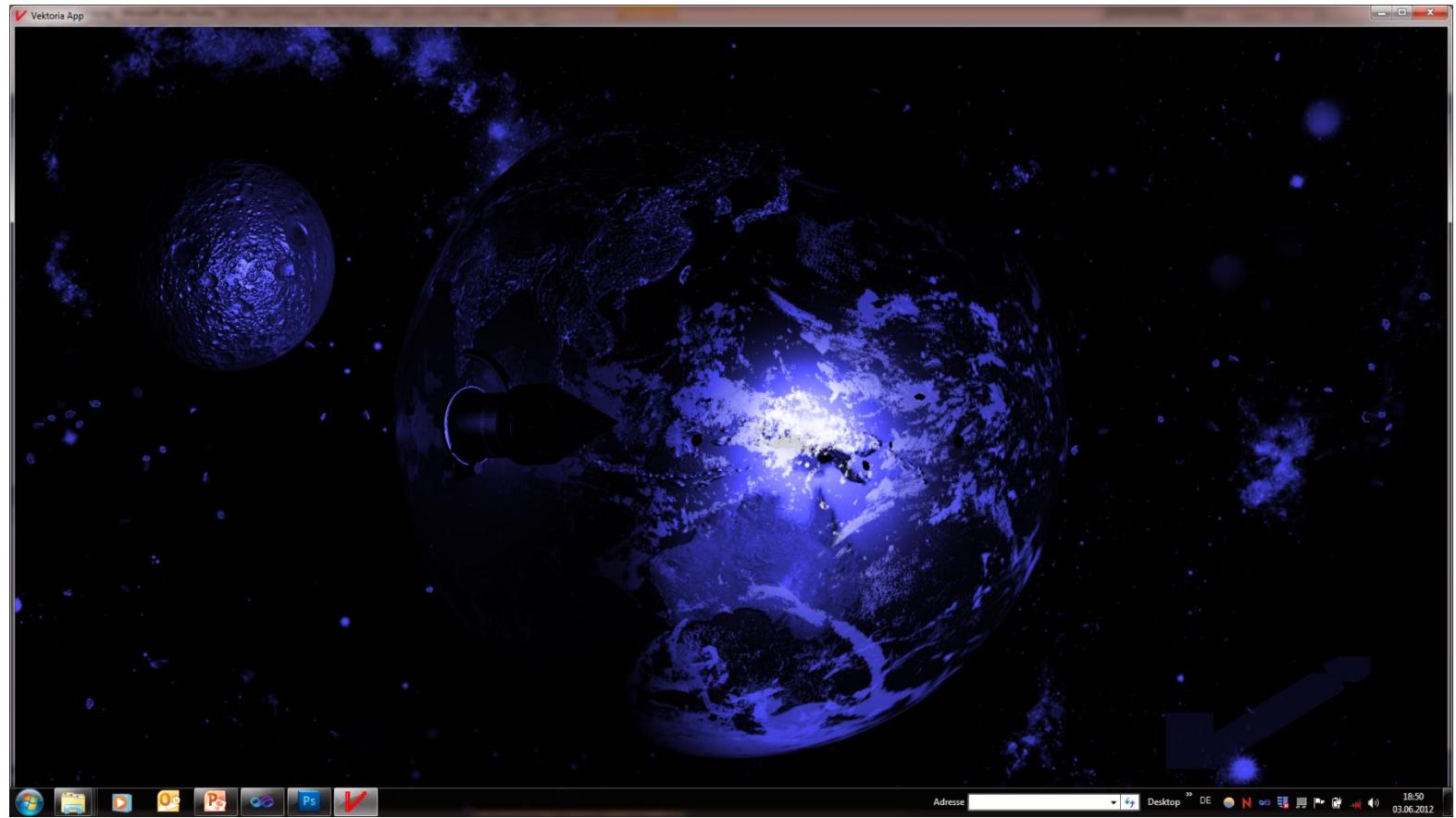
Viewportstile

StylePopArt



Viewportstile

StyleMonochrome (0.3,0.3,1.0)



1 // / / /

2 // / / /

3 // / / /

4 // / / /

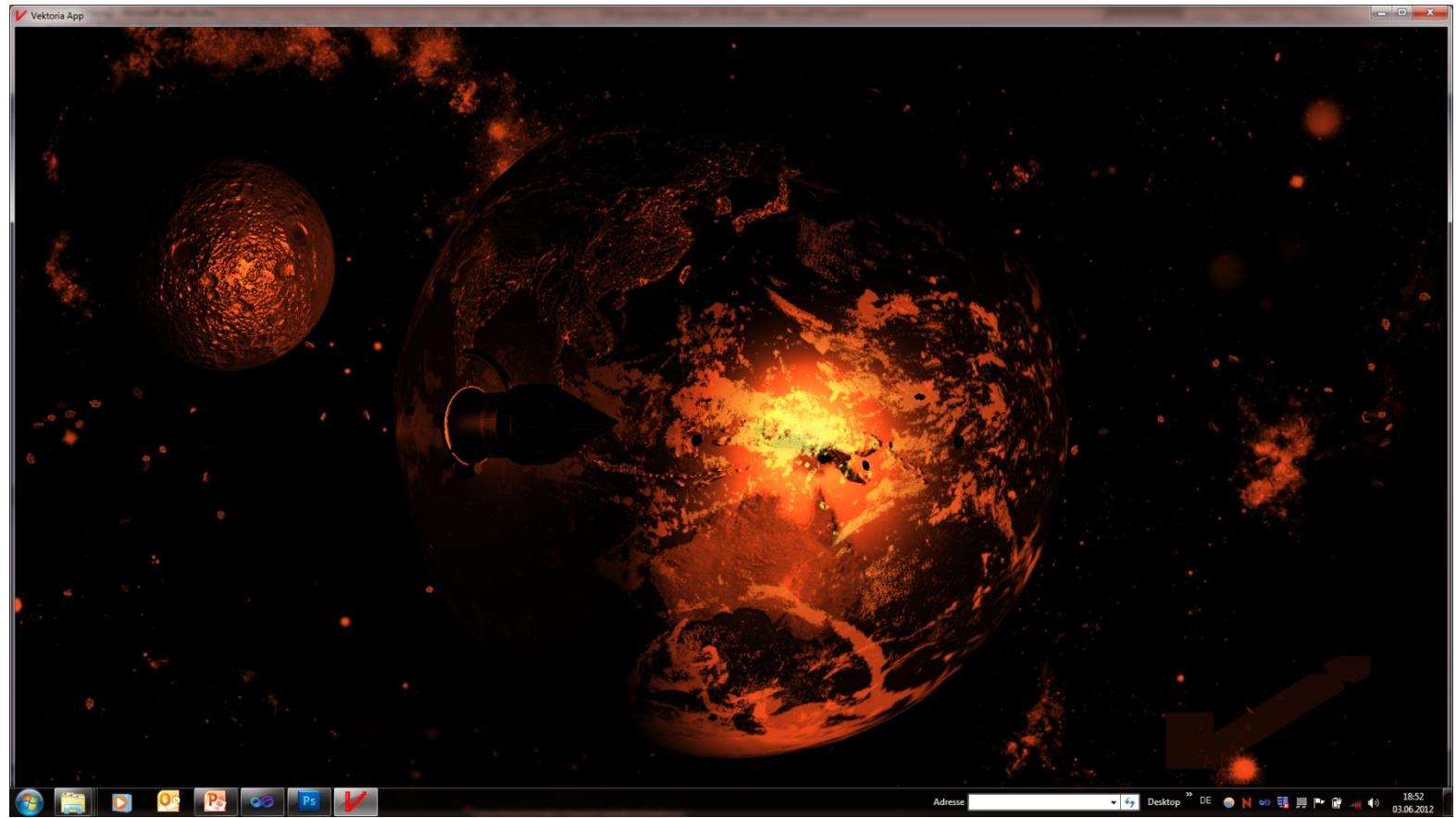
5 // / / /



// / /

Viewportstile

StyleMonochrome (1.0,0.3,0.1)



1 // / / /

2 // / / /

3 // / / /

4 // / / /

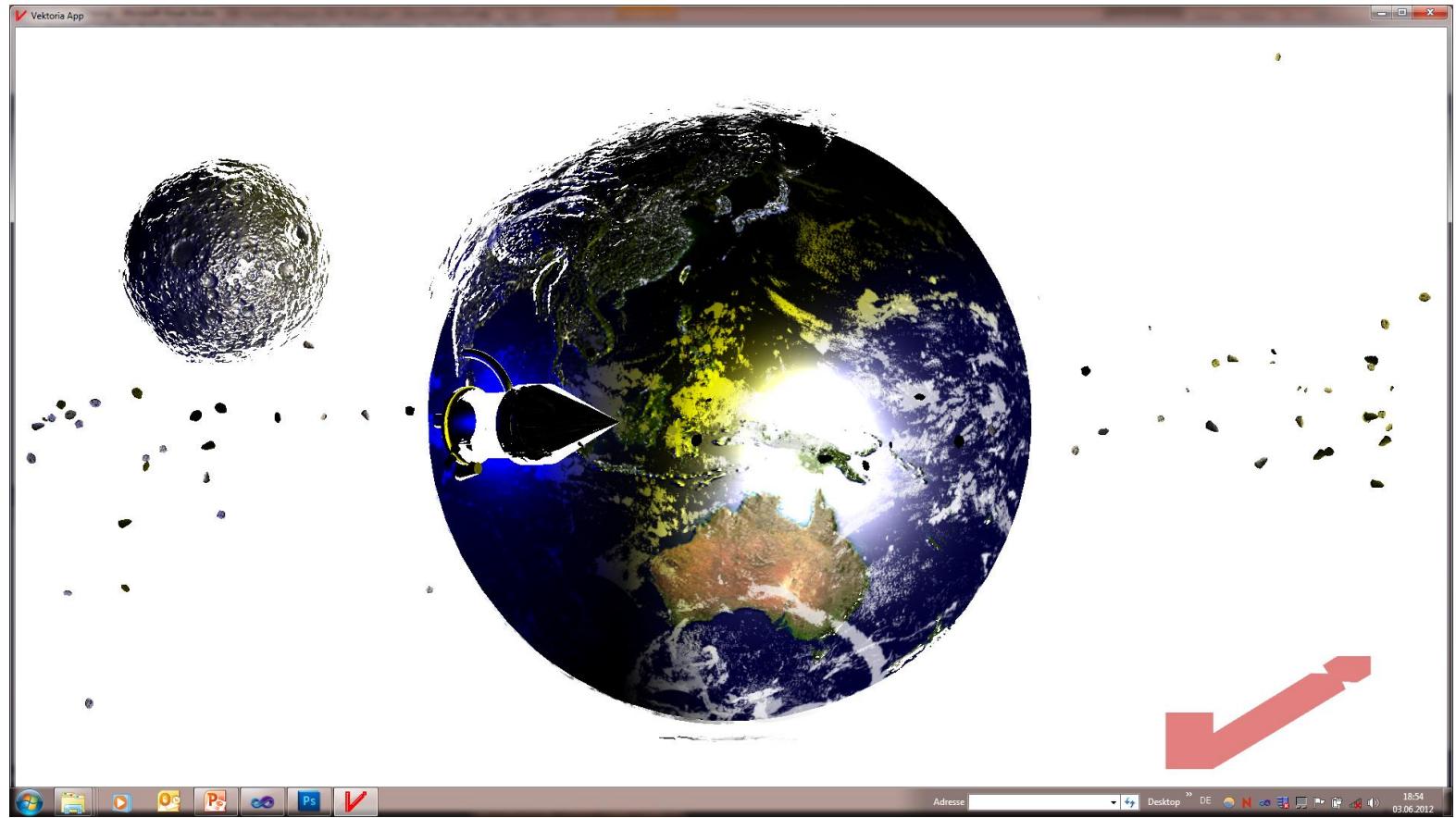
5 // / / /



// / /

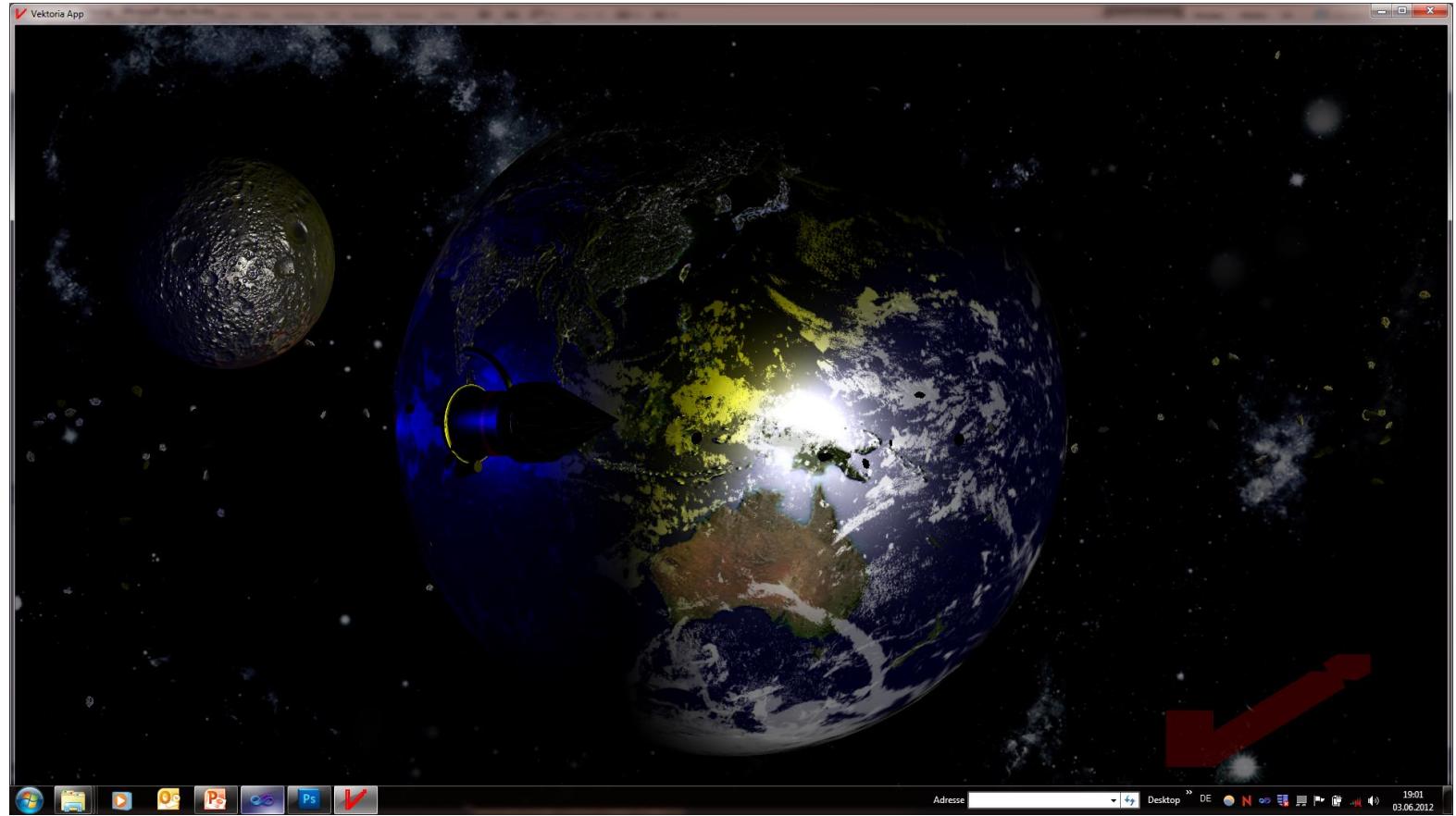
Viewportstile

StyleOutlining



Viewportstile

StyleBrightness (0.5)



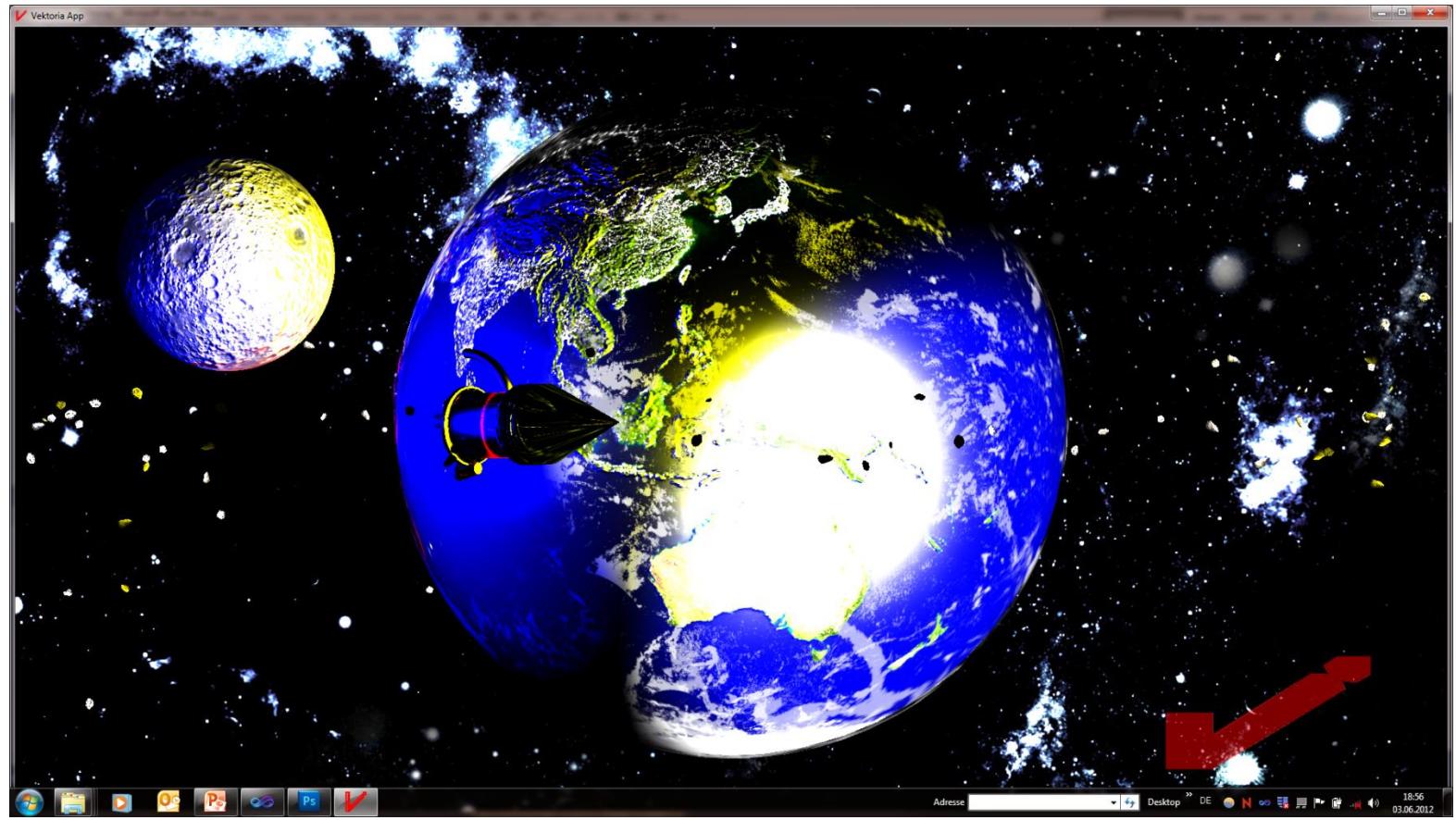
Viewportstile

StyleBrightness (1.0)



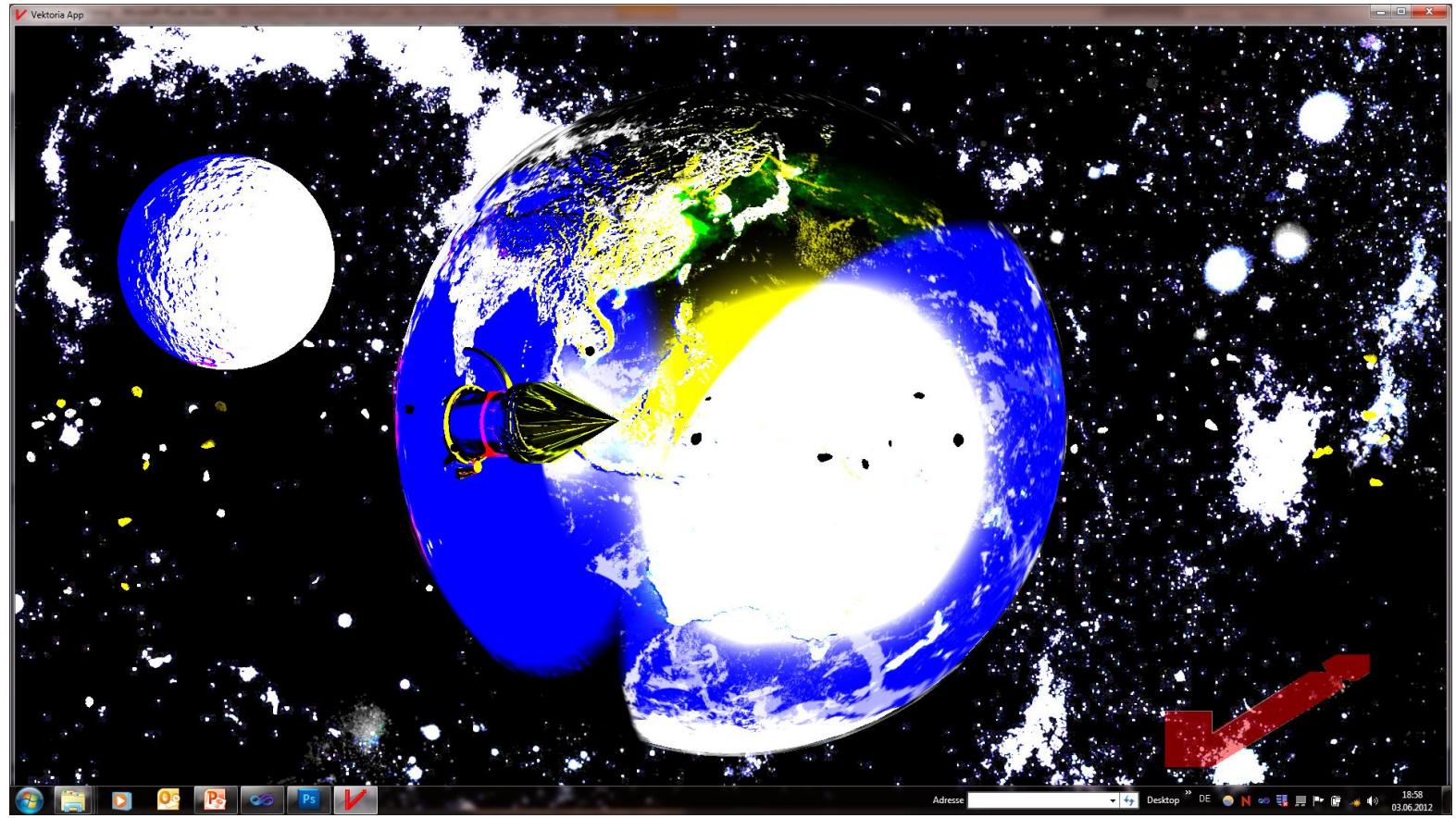
Viewportstile

StyleBrightness (5.0)



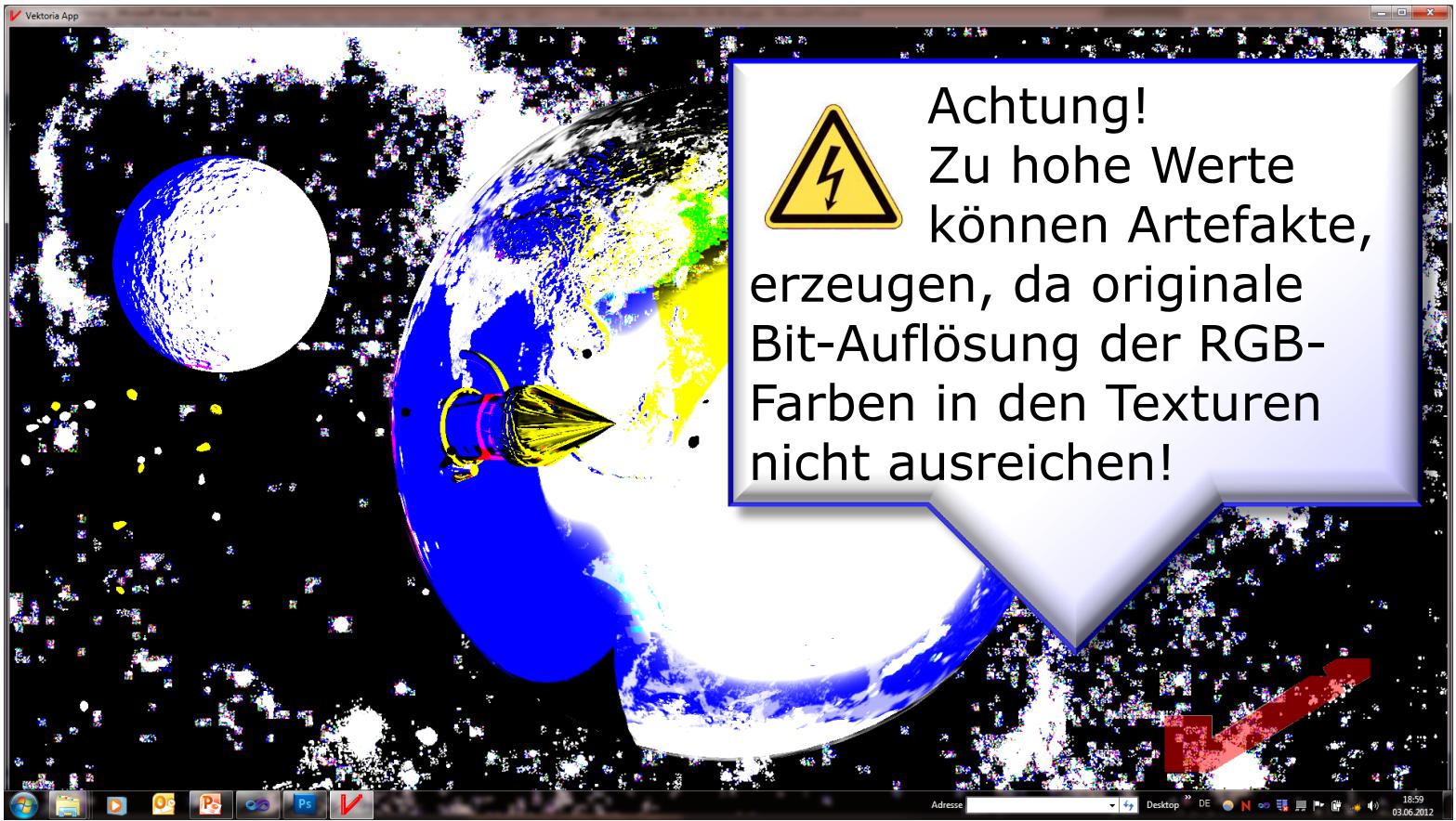
Viewportstile

StyleBrightness (50.0)



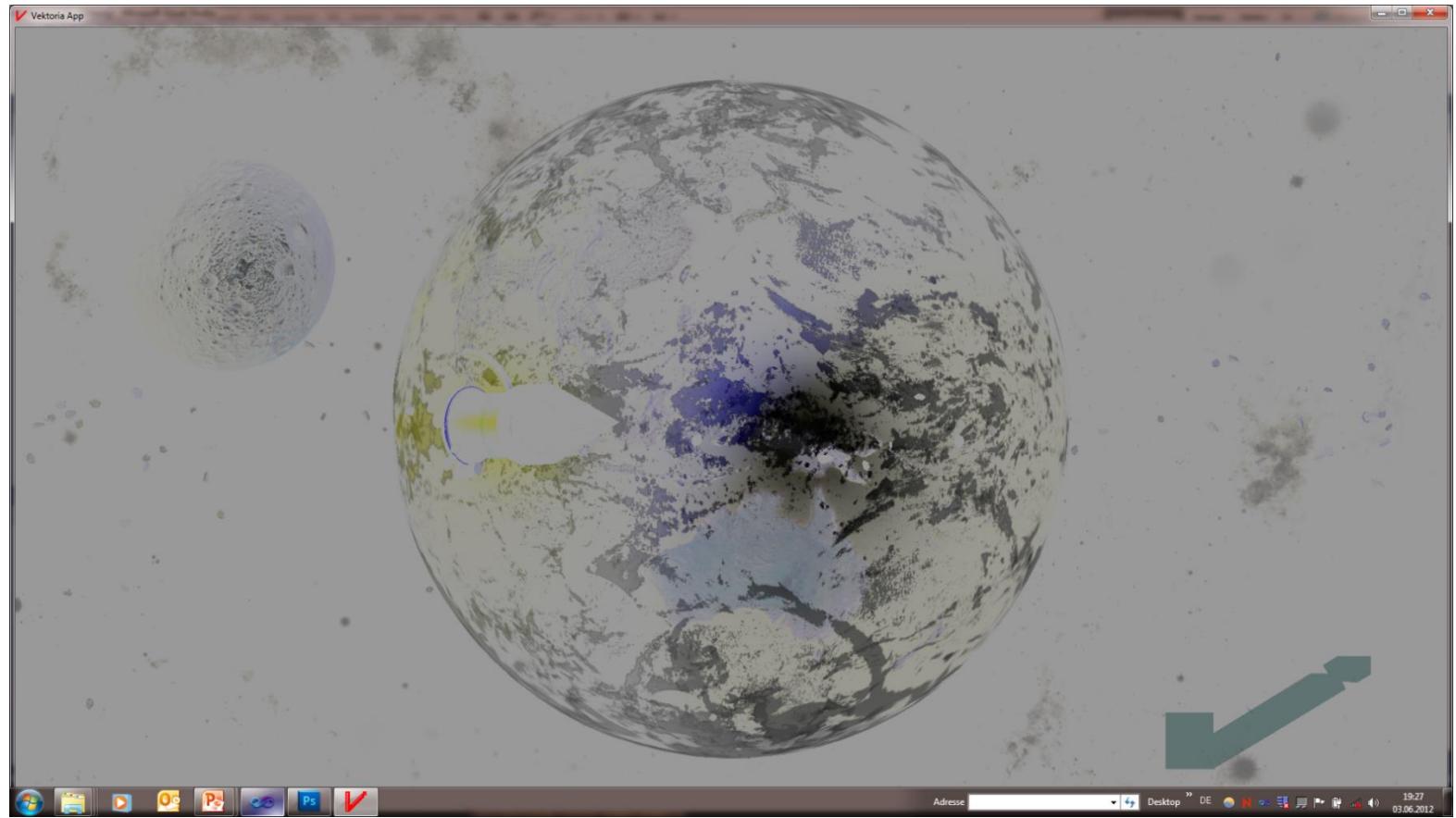
Viewportstile

StyleBrightness (500)



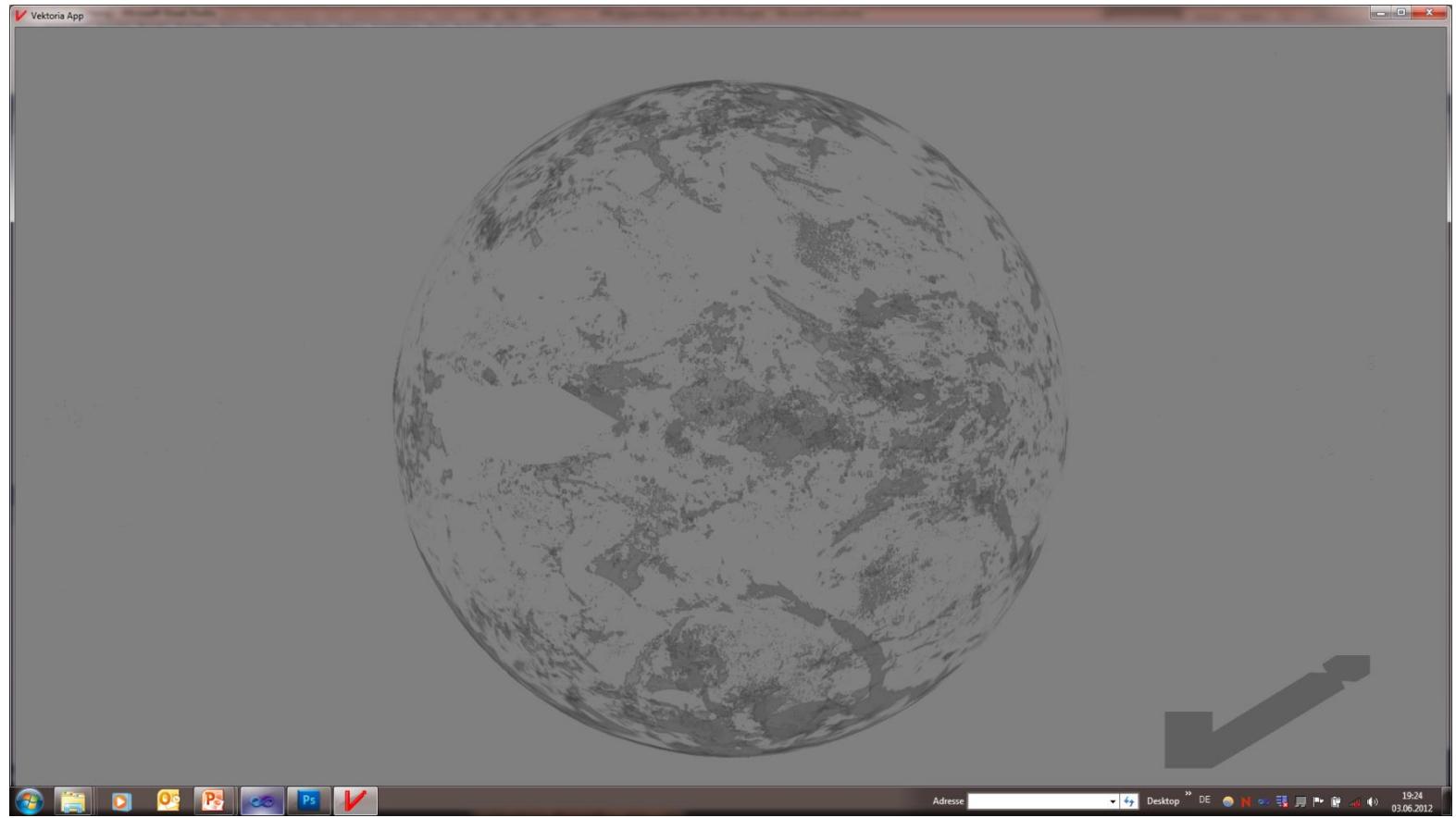
Viewportstile

StyleContrast (-0.6)



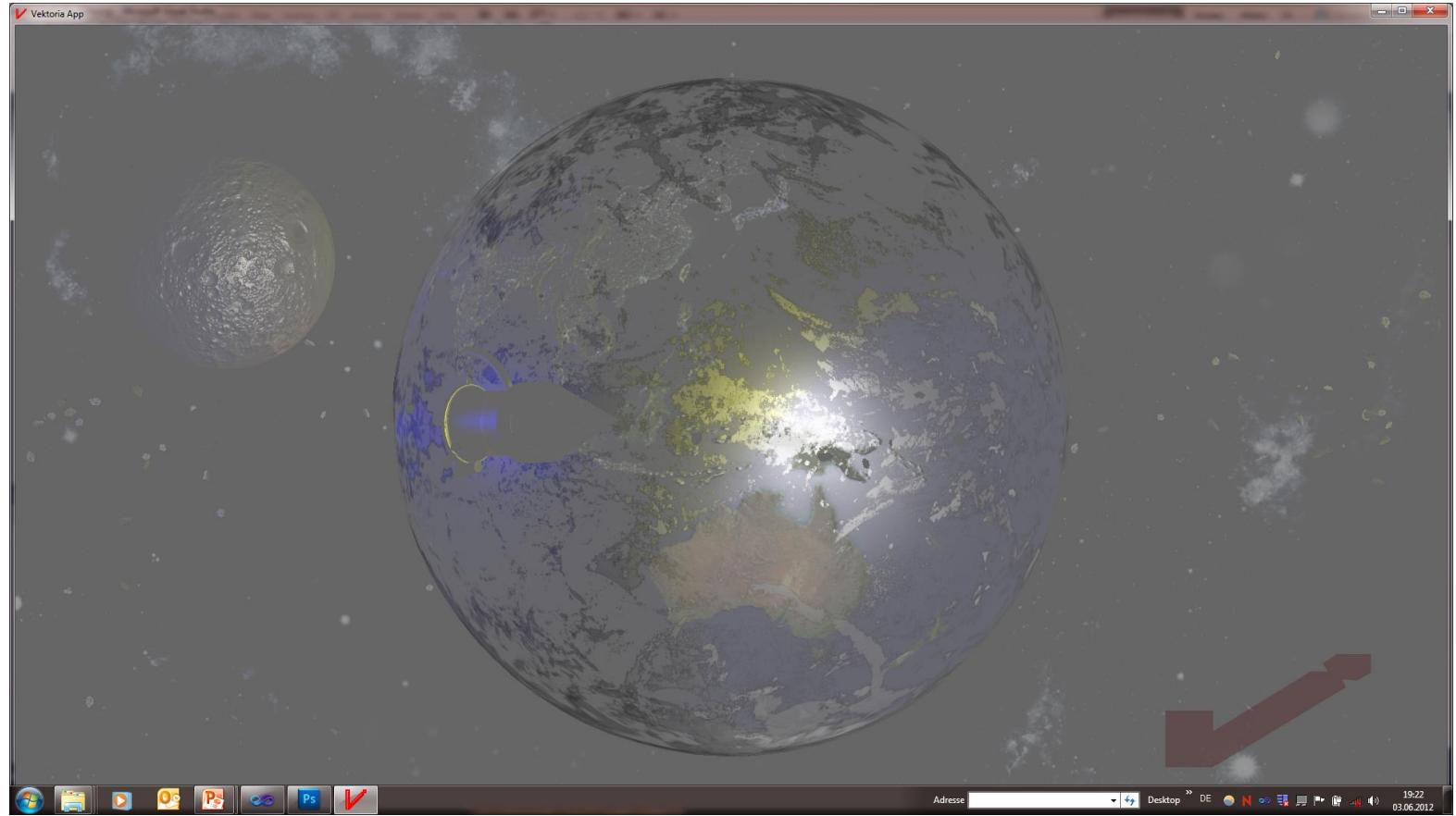
Viewportstile

StyleContrast (-0.5)



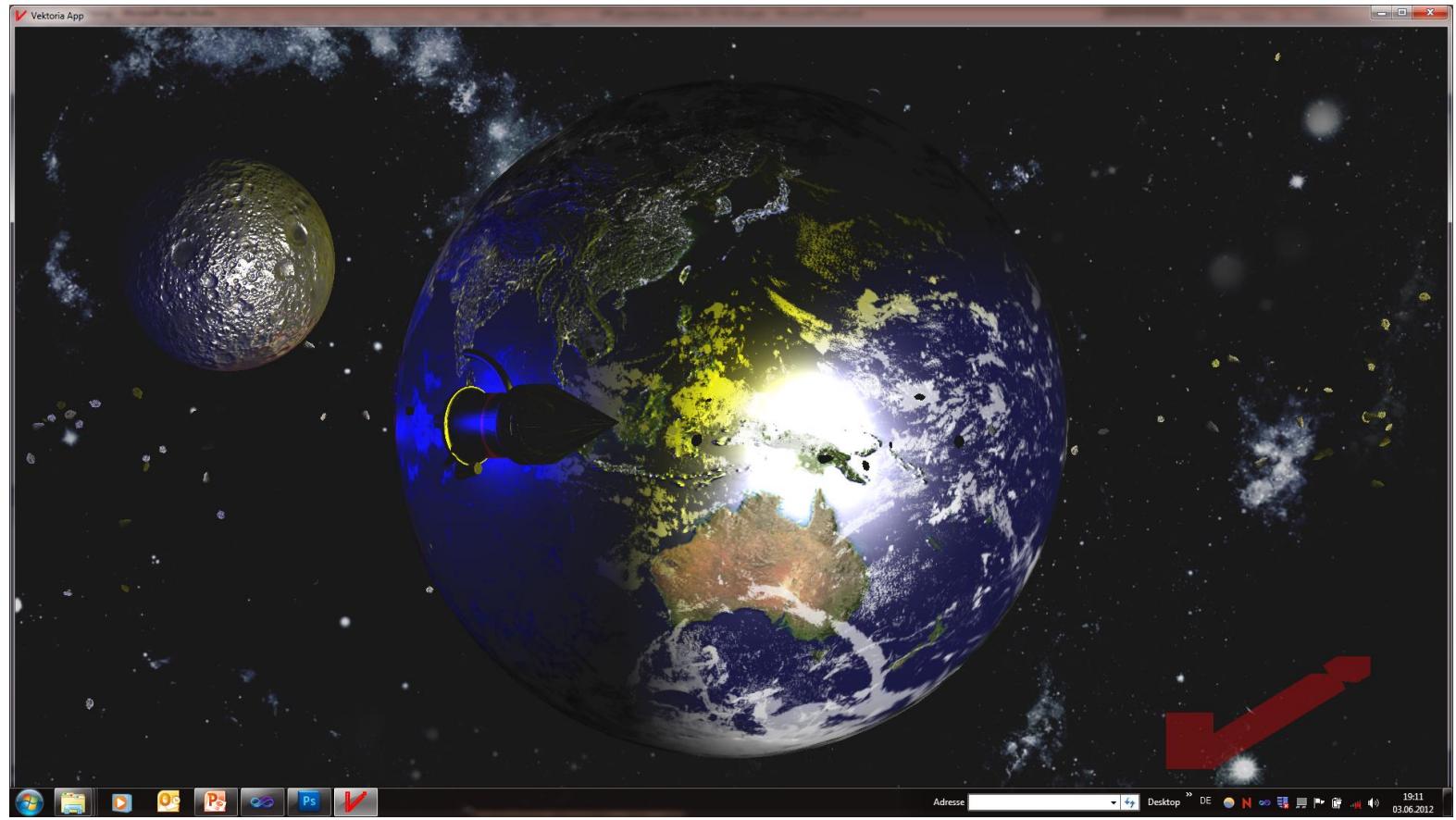
Viewportstile

StyleContrast (-0.1)



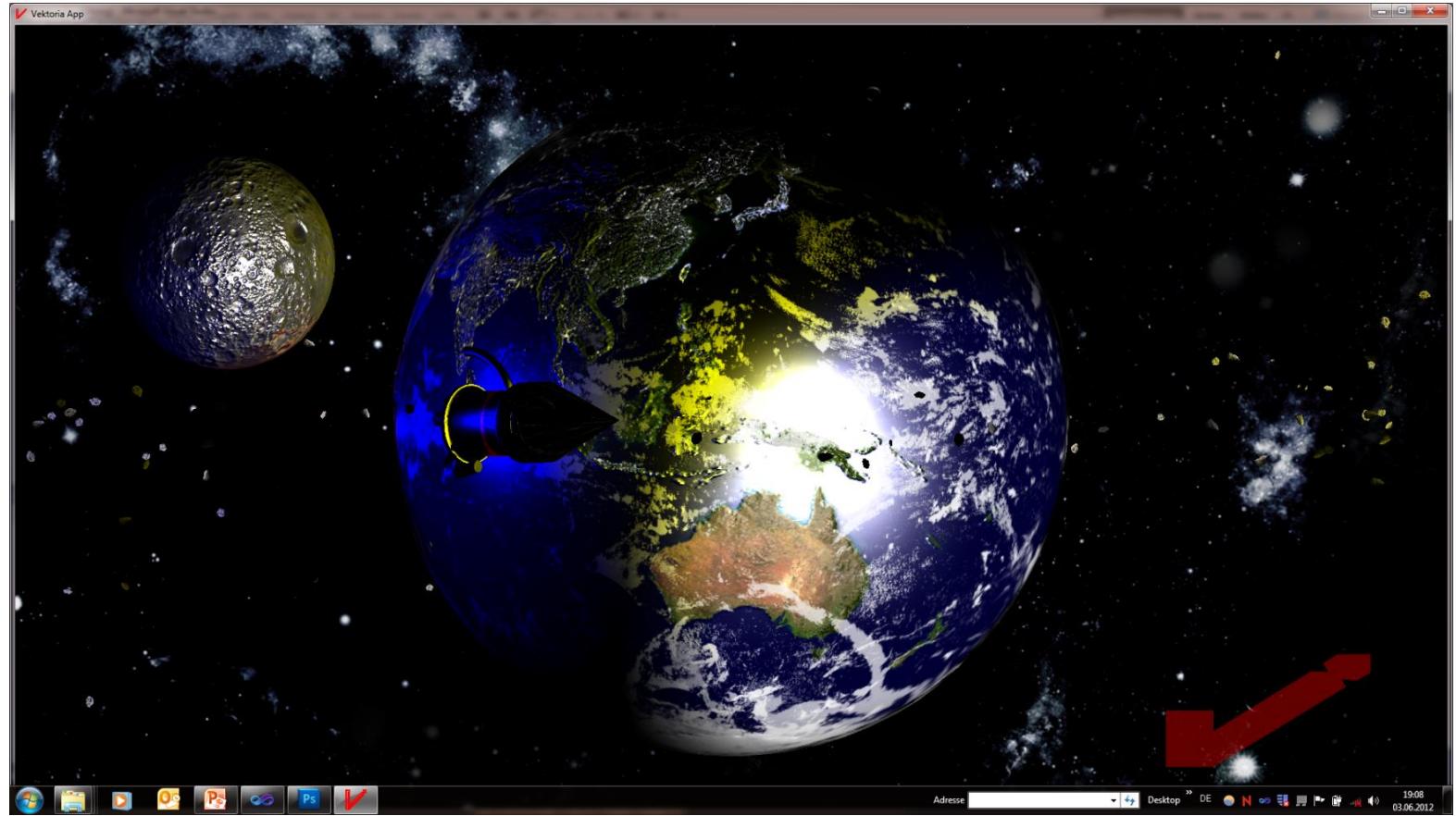
Viewportstile

StyleContrast (-0.1)



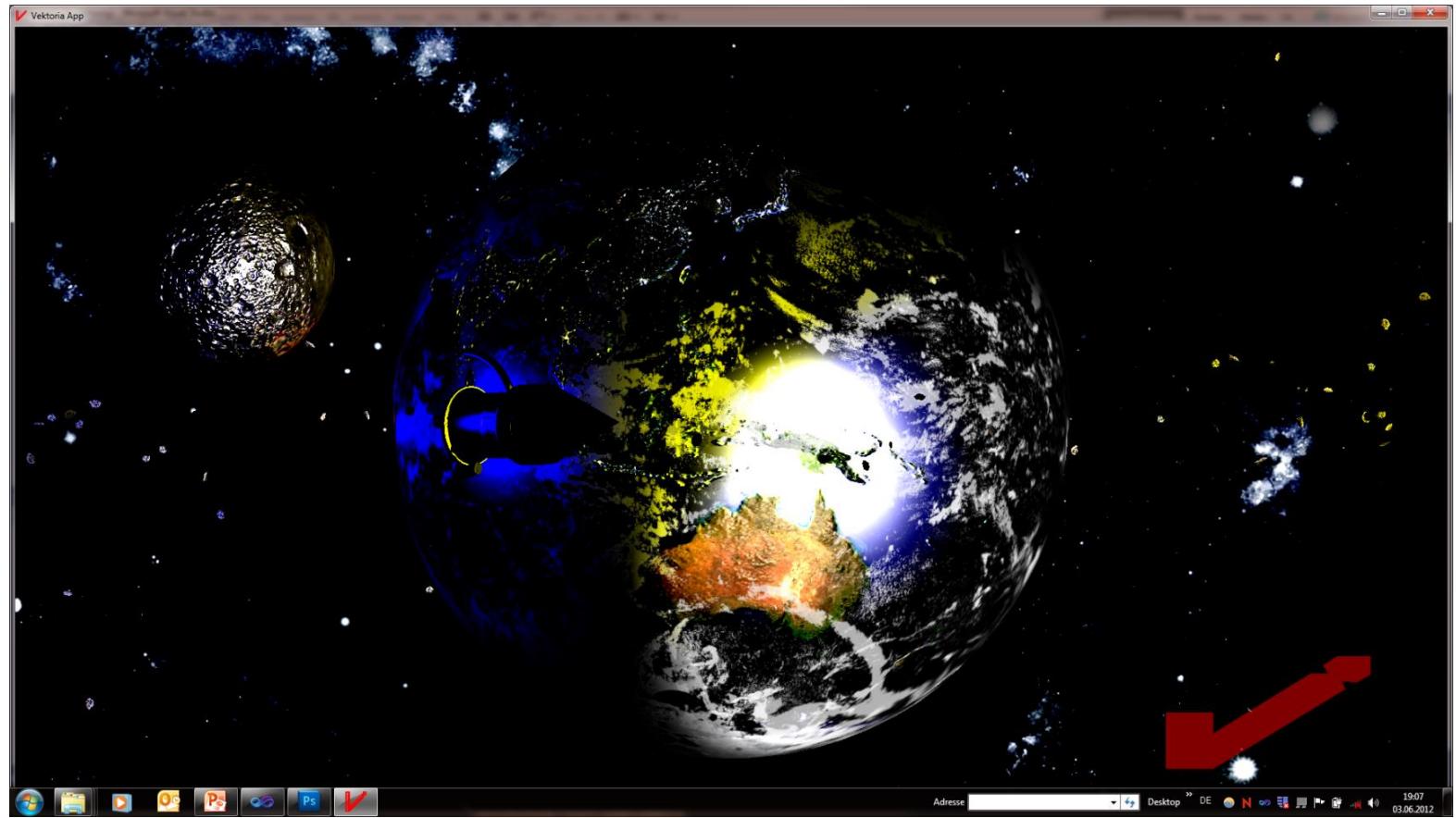
Viewportstile

StyleContrast (0.0)



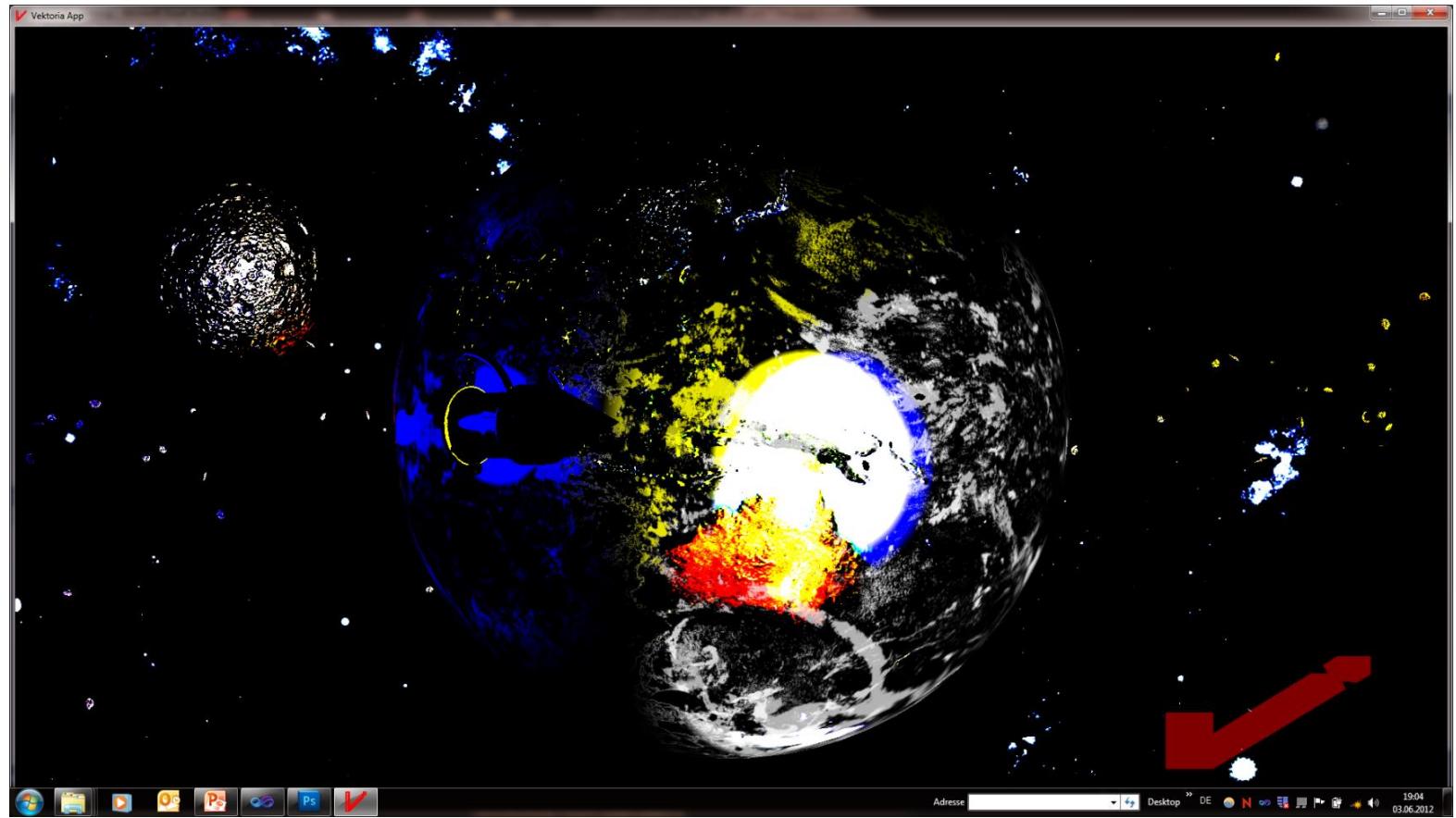
Viewportstile

StyleContrast (0.5)



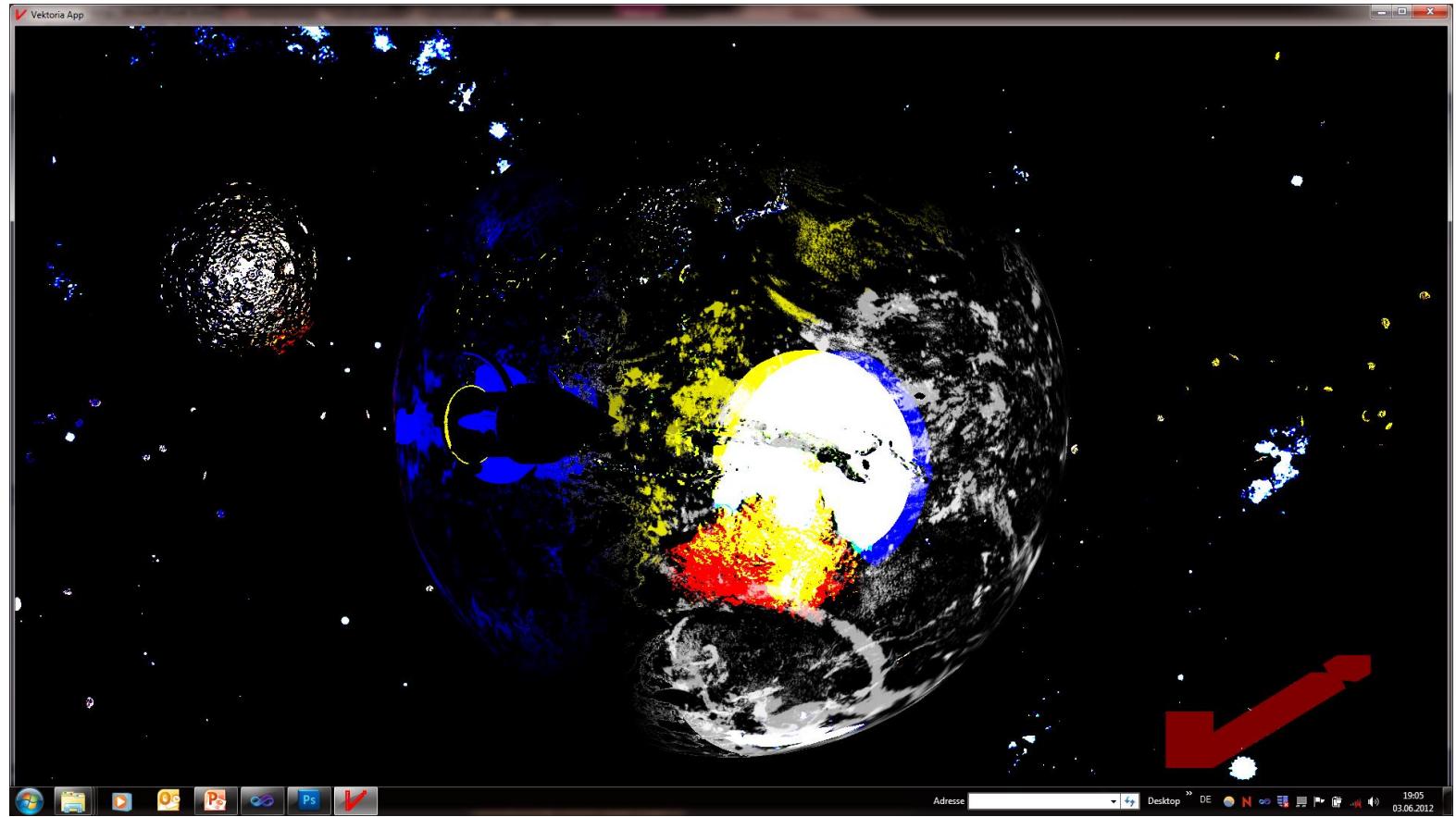
Viewportstile

StyleContrast (5.0)



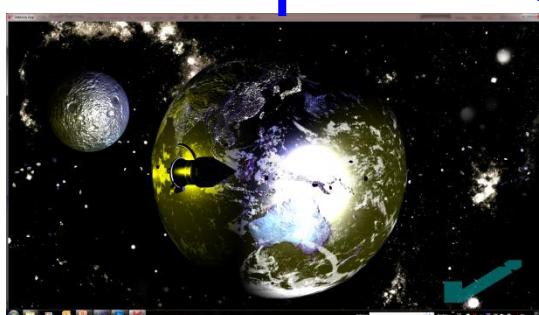
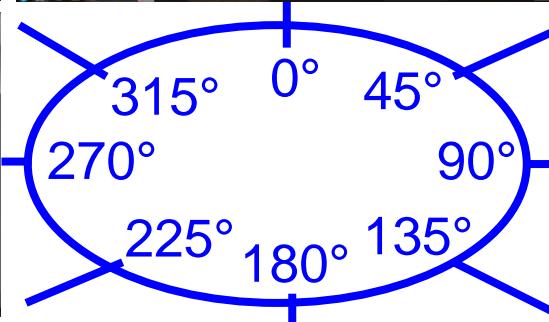
Viewportstile

StyleContrast (500.0)



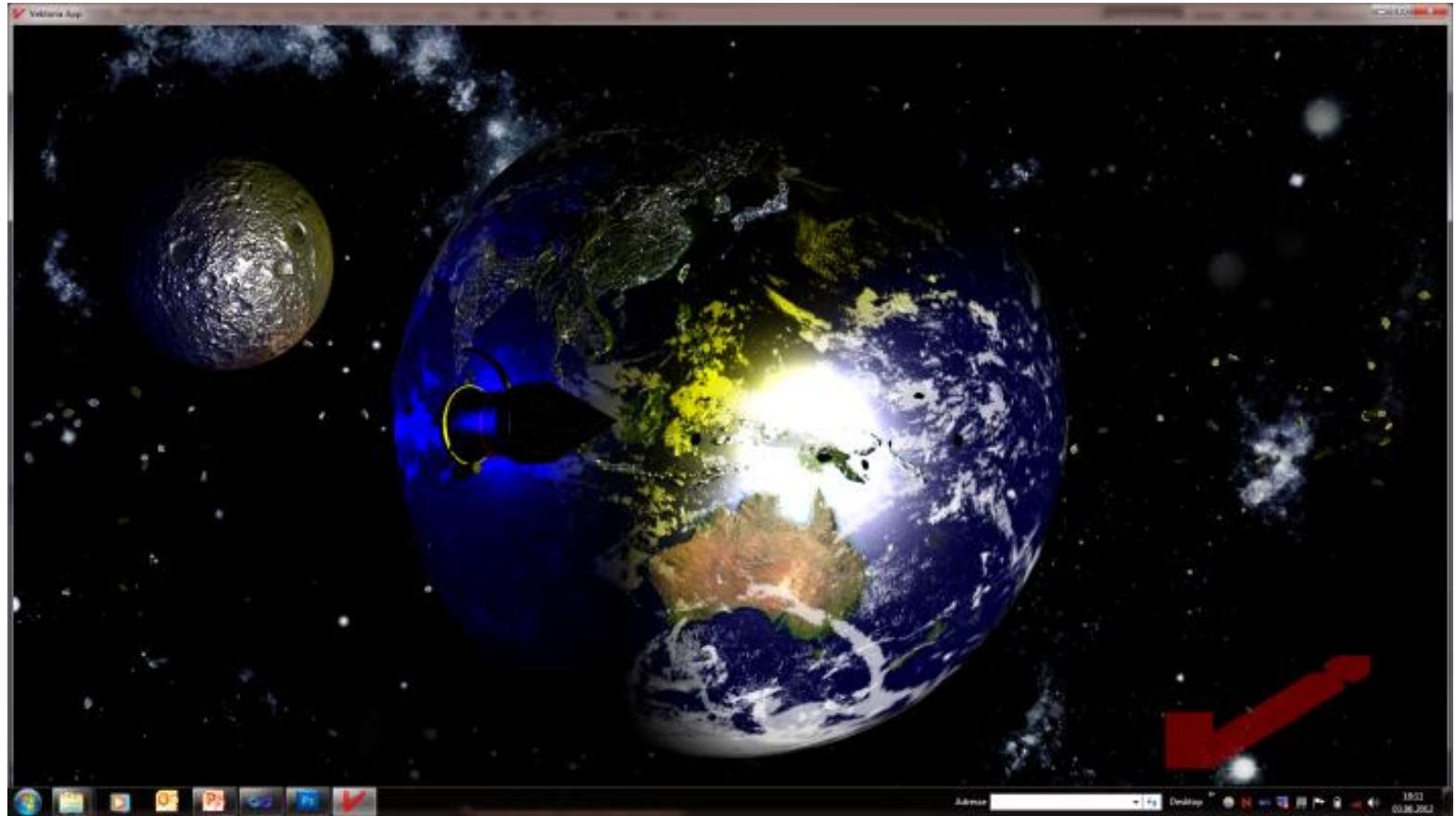
Viewportstile

StyleRotateHue



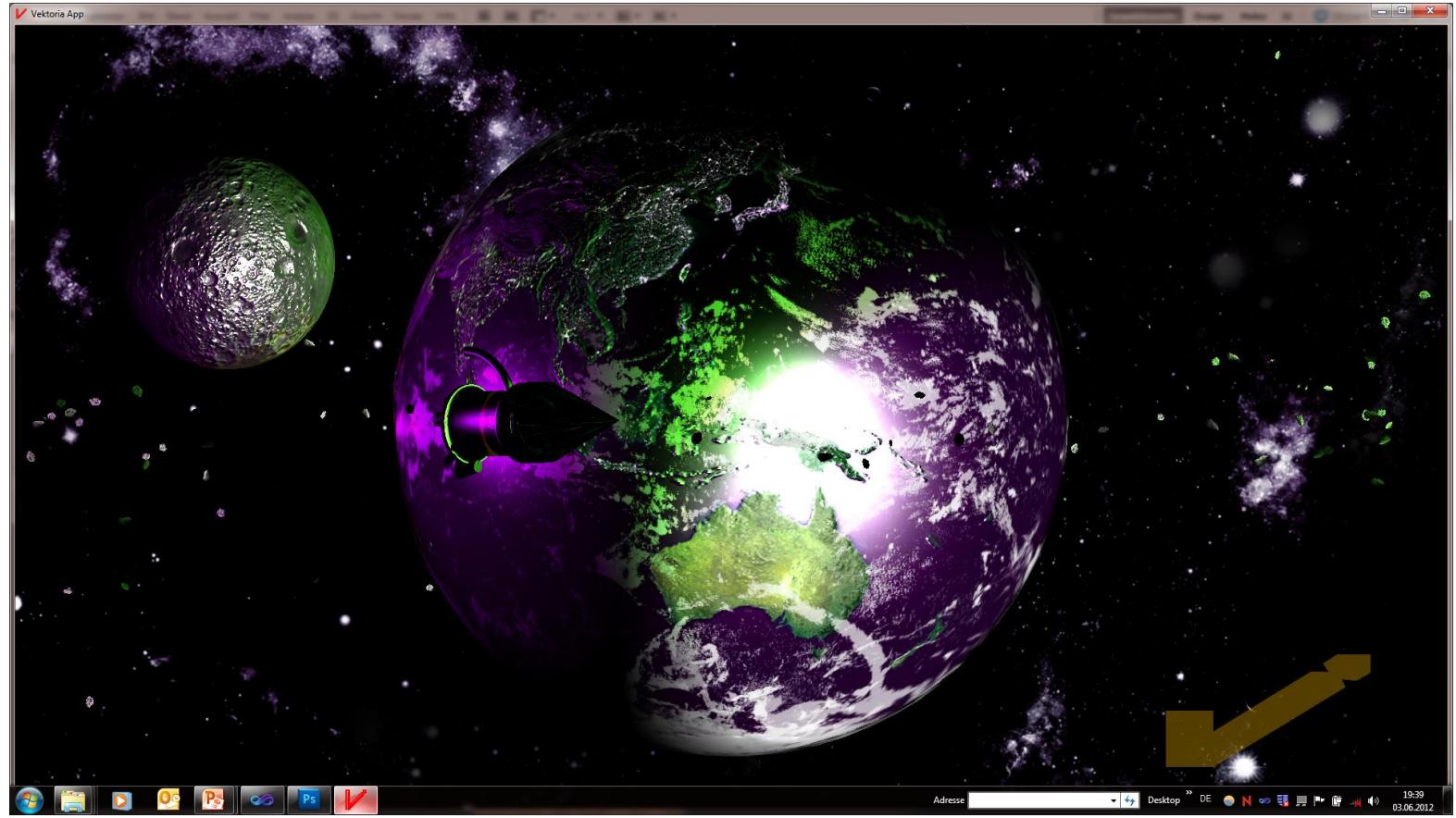
Viewportstile

StyleRotateHue(0)



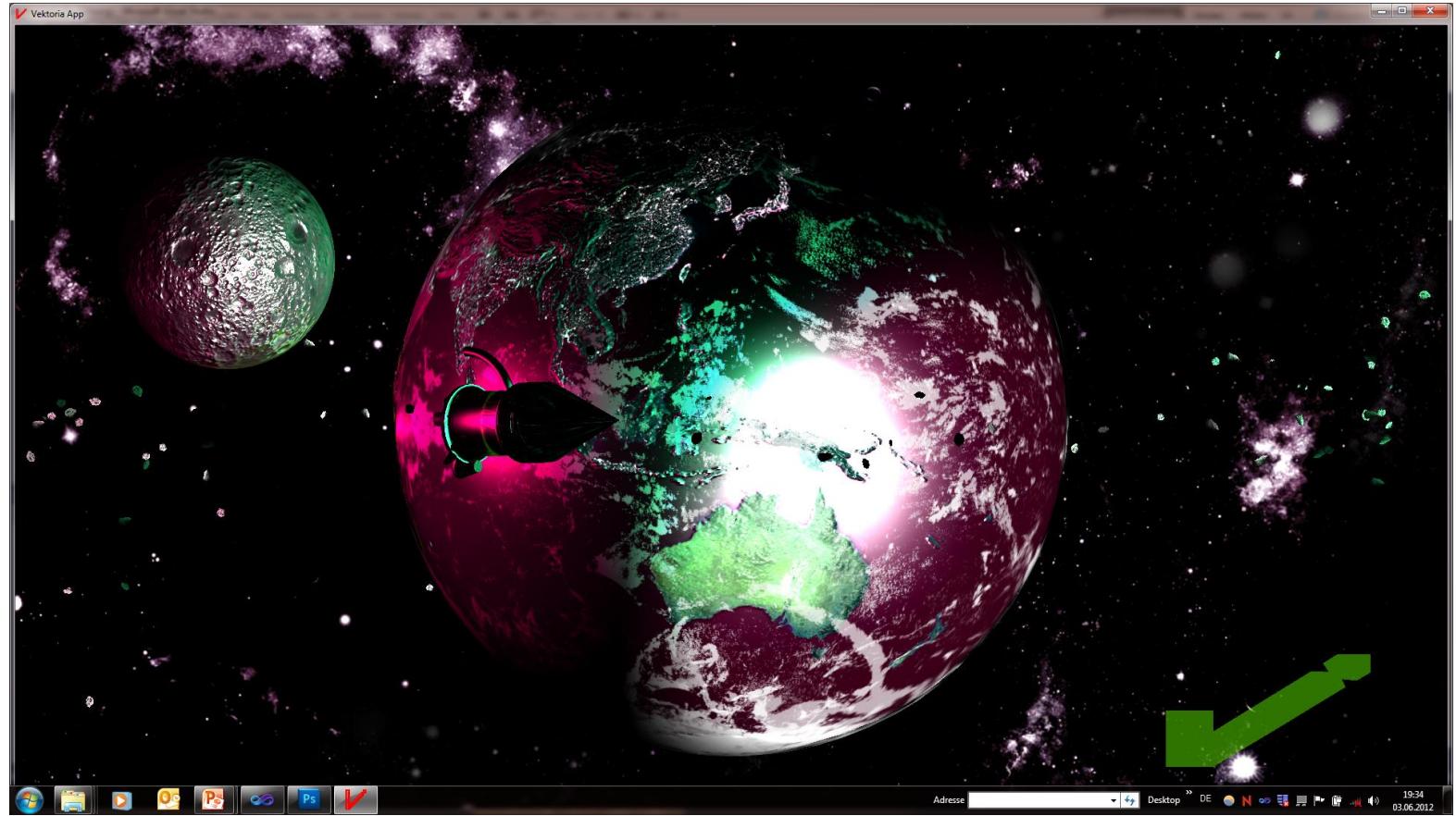
Viewportstile

StyleRotateHue(QUARTERPI)



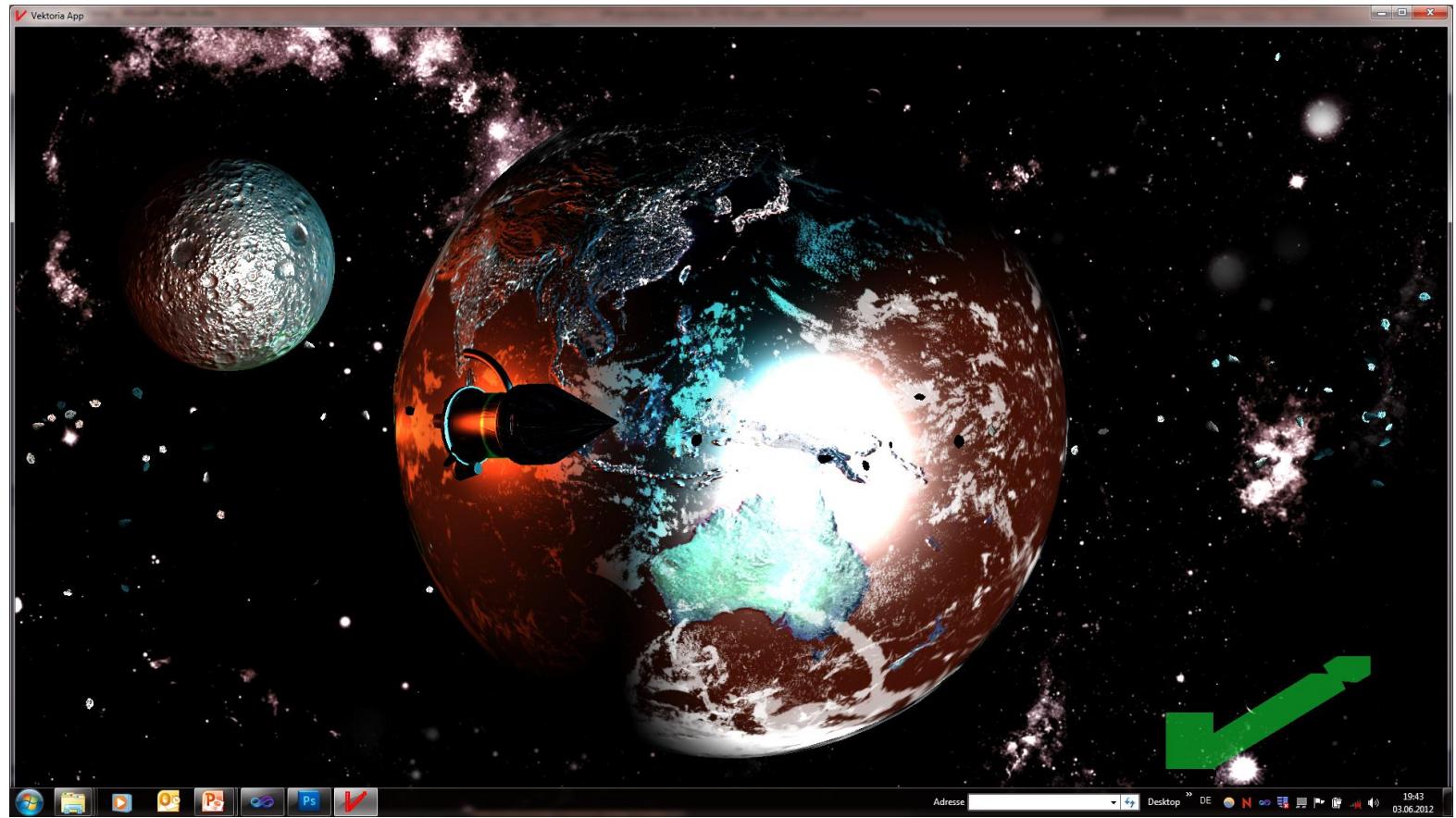
Viewportstile

StyleRotateHue(HALFPI)



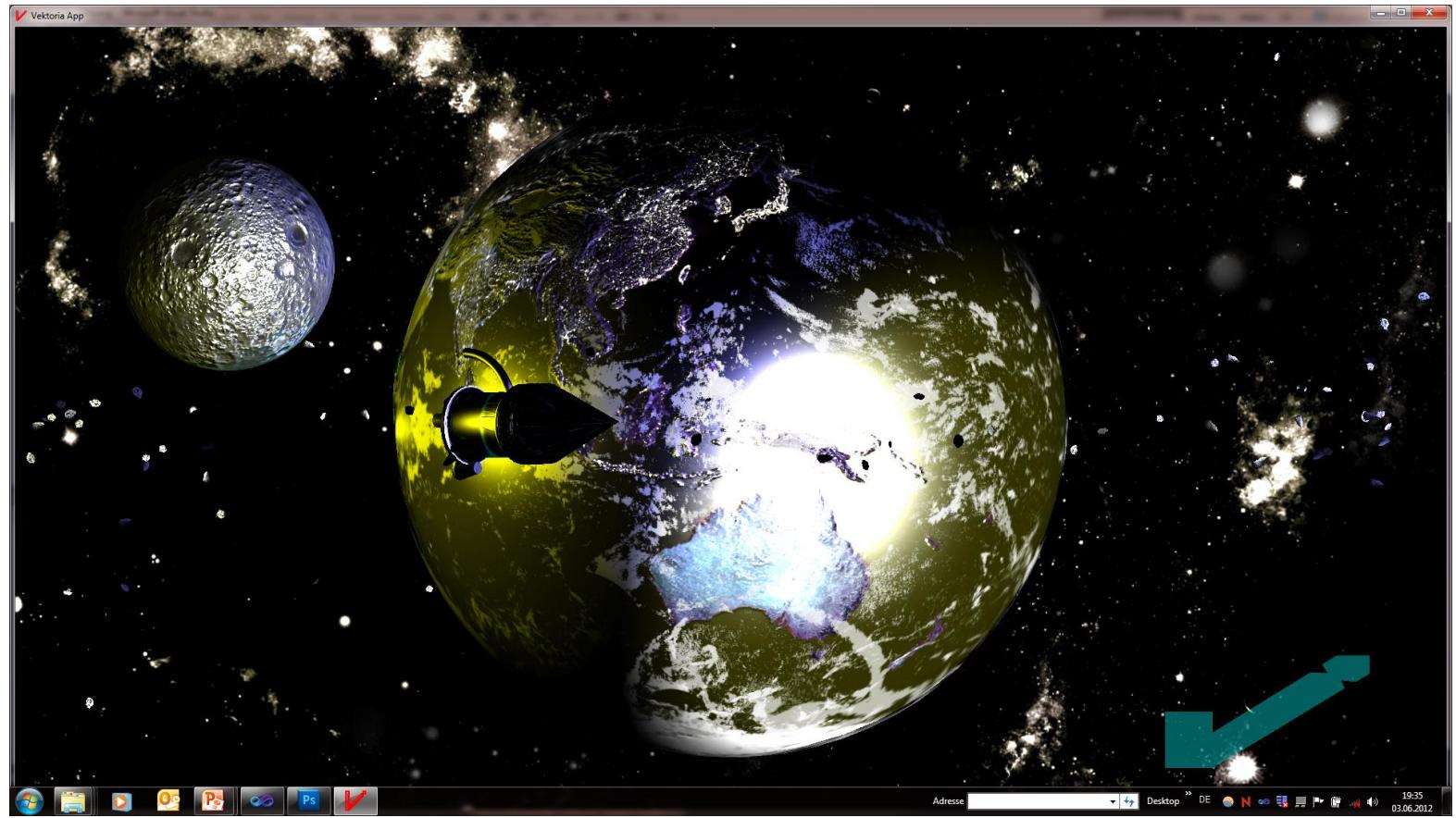
Viewportstile

StyleRotateHue($\pi * 0.75$)



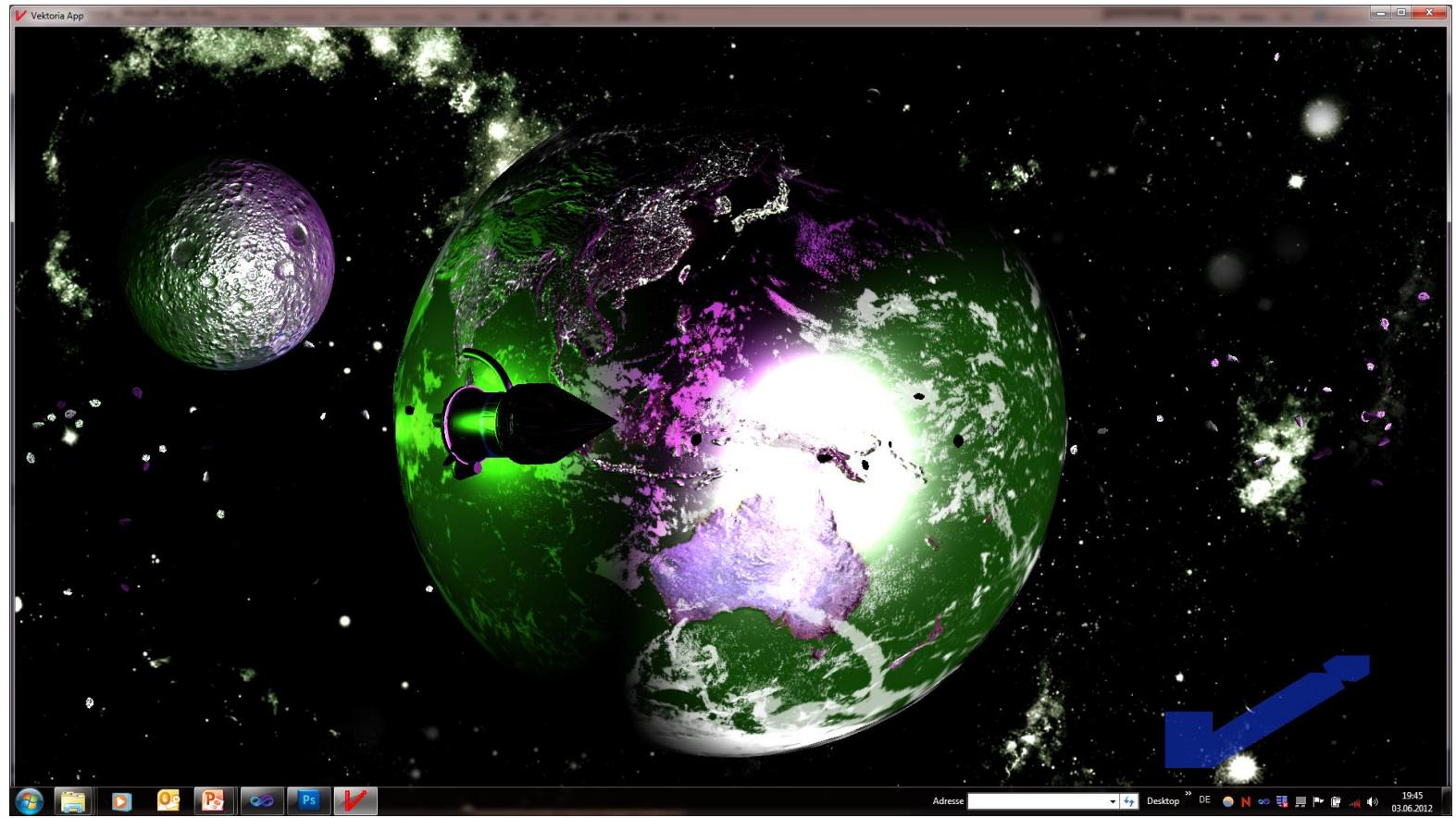
Viewportstile

StyleRotateHue(PI)



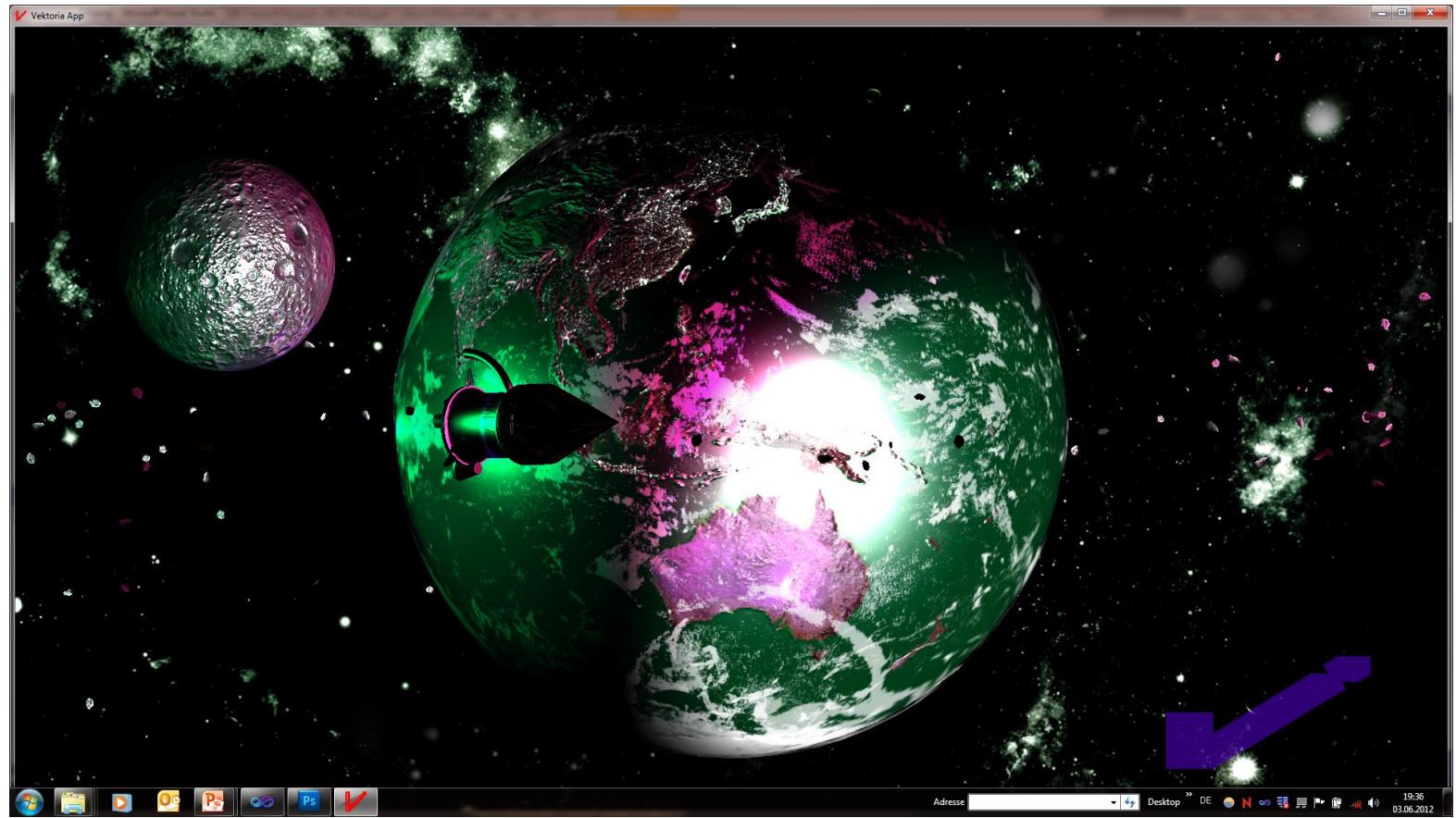
Viewportstile

StyleRotateHue(PI*1.25)



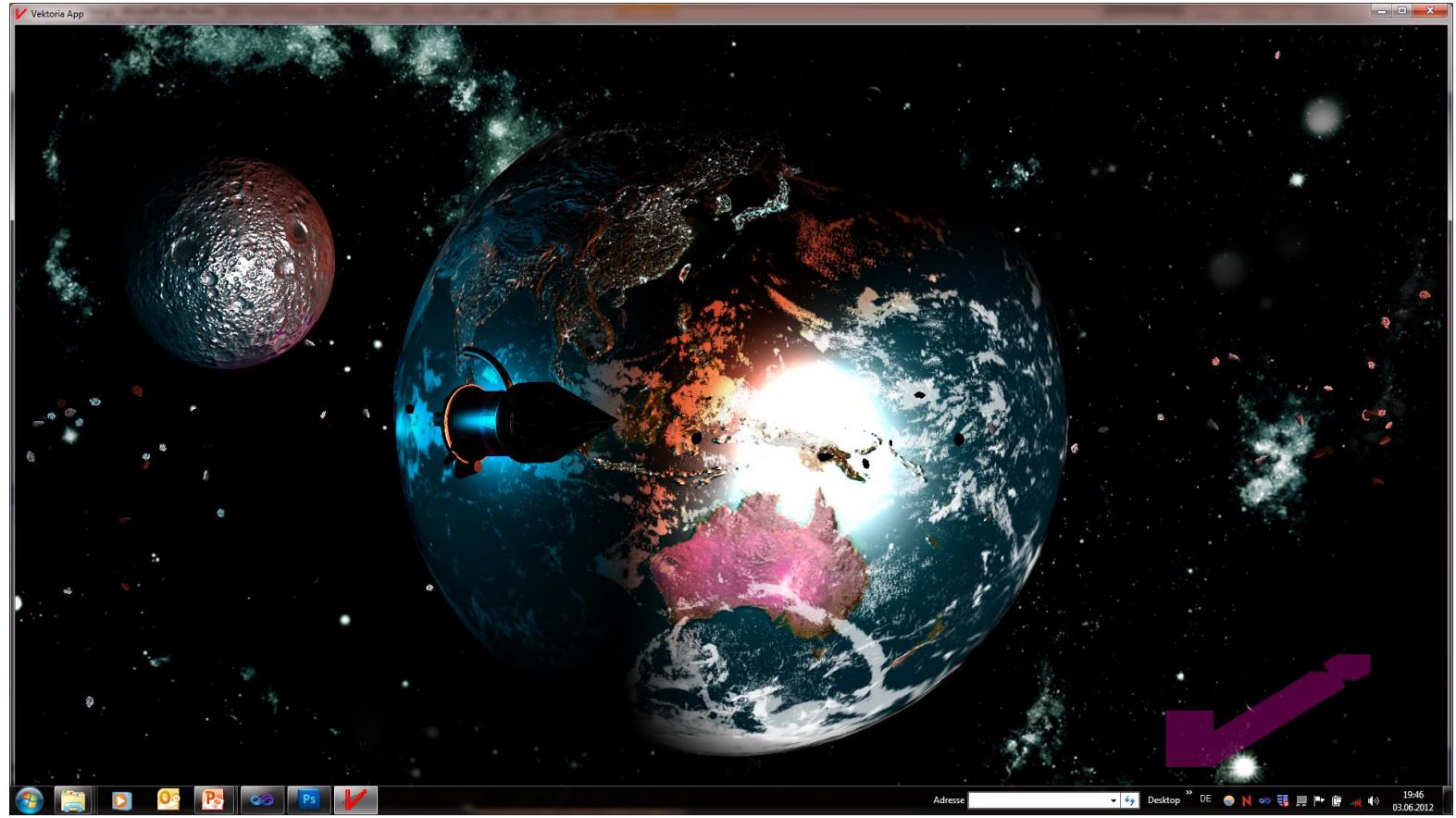
Viewportstile

StyleRotateHue(PI+HALFPI)



Viewportstile

StyleRotateHue(PI*1.75)



Viewportstile

StyleRotateHue(TWOPI)





Viewportstile

Übung zu Viewportstilen



Probieren Sie verschiedene Stile aus!



Freiwillige Zusatzaufgabe für die schnellen Nerds:

- Rotieren Sie während der Laufzeit die Farbgebung der Szene!



StyleOwn()

1 //
Mit der Methode StyleOwn der Klasse CViewport
lässt sich ein eigener Shader-Teil in HLSL
programmieren.

2 //
Der Punkt, in dem man seinen eigenen Code
einpflegen kann, ist in der Datei „shader.hlsl“
markiert.

3 //
Achtung, um StyleOwn zu benutzen muss im
Compiler #

4 //
5 //



Shadersprachen

- 1 // / / / ● **GLSL** (Aussprache: „tschielsläng“)
OpenGL Shading Language
OpenGL-spezifischer C-Dialekt
- 2 // / / / ● **Cg**
C for Graphics
- 3 // / / / Von NVidia entwickelter C-Dialekt, der auch von anderen Grafikkartenherstellern unterstützt wird. Ist sowohl für OpenGL als auch Direct3D einsetzbar
- 4 // / / / ● **HLSL**
High Level Shading Language:
Microsofts Implementierung von Cg, ist in Direct3D enthalten
- 5 // / / /



Gg Profile

- GPU-Programmierung ist noch nicht auf der selben Abstraktionsstufe wie die CPU-Programmierung angelangt.
- Jeder Grafikprozessor hat stark unterschiedliche Fähigkeiten
- Die Basis-APIs haben unterschiedliche Schnittstellenmöglichkeiten
- ⇒ Lösung: Profilangabe
- ⇒ Ein Cg-Profil ist eine Untermenge von Gg



Gg Profile



	Basis-API	VS-Profil	FS-Profil
1	OpenGL ARB	arbvp1	arbfp1
2	OpenGL NV40	vp40	fp40
3	OpenGL NV30	vp30	fp30
4	OpenGL NV2X	vp20	fp20
5	DirectX 11.0	vs_5_x	ps_5_x
	DirectX 10.0	vs_4_x	ps_4_x
	DirectX 9.0	vs_2_x	ps_2_x
	DirectX 8.0	vs_1_1	ps_1_1, ps_1_2, ps_1_3

Cg/HLSL elementare Datentypen

1	• bool	Wahrheitswert
2	• int	32bit Ganzzahl
3	• uint	32bit unsigned int (nur bei HLSL)
4	• fixed	12bit Fixkommazahl (1-1-10)
5	• half	16bit Gleitkommazahl
	• float	32bit Gleitkommazahl
	• double	64bit Gleitkommazahl
	• float2	2D-Gleitkommavektor
	• float3	3D-Gleitkommavektor
	• float4	4D-Gleitkommavektor
	• matrix	Homogene 4x4-Matrix (nur bei HLSL)
	• float4x4	Homogene 4x4-Matrix (nur bei Cg)



Cg/HLSL komplexe Datentypen

1 // / / /

- sampler

2D-Texturobjekt (nur bei Cg)

2 // / / /

- Texture1D

1D-Texturobjekt (nur bei HLSL)

3 // / / /

- Texture2D

2D-Texturobjekt (nur bei HLSL)

4 // / / /

- Texture3D

3D-Texturobjekt (nur bei HLSL)

5 // / / /

- struct

Union

6 // / / /

- []

Array

7 // / / /

- [][]...

multidim. Array

8 // / / /

- vector<datent, i>

Vektor aus i Variablen des Datentyps d
(nur bei HLSL)

9 // / / /

Beispiele:

- vector<uint, 4>

Vektor aus 4 unsigned integers (HLSL)

- vector<float, 8>

Vektor aus 8 floats (HLSL)



Cg-Parameterarten (Beispiele)

1 // Eingabeparameter

- `in float3 i_f3Pos : POSITION;`
- `in float4 i_f4Color : COLOR;`

2 // Ausgabeparameter

- `out float3 o_f3Pos : POSITION;`
- `out float4 o_f4Color : COLOR;`

3 // Konstante Eingabeparameter

- `uniform float fTime;`
- `uniform float4x4 f44Rotation;`



Mathematische Cg/HLSL-Funktionen

- 1 // / / / • mul (A,B) Multiplikation von Vektoren und Matrizen
 (überladene Funktion)
- 2 // / / / • abs(A) Absolutfunktion
- 2 // / / / • log2(A) Zweierlogarithmus
- 2 // / / / • sqrt(A) Wurzel
- 2 // / / / • pow(A,B) Potenzfunktion
- 3 // / / / • saturate(A) Schneidet Wert ab,
 so dass er zwischen 0 und 1 liegt (nur in HLSL)
- 4 // / / / • ...

Kein #include notwendig!



Cg/HLSL-Texturfunktionen

- Sample (*texture2D, float2*) (Nur in HLSL)
- texCUBE (Nur in Cg)
- tex3Dproj (3D-Textur-> 2D)
- *texture2D.GetDimensions(Bittiefe,Breite,Höhe, Ebenen)*
- ...

Kein #include notwendig!



Cg/HLSL-Vektorfunktionen

- `dot(V,V)`
- `normalize(V)`
- `mul(V,M)`
- ...

Dot-Produkt
Vektornormalisierung
Vektor-Matrix-Multiplikation

Ansonsten normale Operatoren verwenden (*, +, /, etc.)

Kein `#include` notwendig!



Was Cg/HLSL mit C gemeinsam hat

- structs
- Arrays, mehrdimensionale Arrays
- arithmetische Operatoren
- Boolsche Operatoren
- Increment- und Decrementoperatoren
- Konditionaloperator
- Zuweisungsoperatoren
- Kommaoperator
- Benutzerdefinierte Funktionen
- Einfacher Verzweigungskonstruktor
- Schleifenkonstrukte
- Abbruchkonstrukten
- Kommentare
- Einige Präprozessorkonstrukte

struct
[], [2][3], ...
+, *, /, %, ...
||, &&, !, ...
++, --
?:
= , +=, *=, /=, %=,...
,

void MyFkt(float f){}

if
do, while,
break, continue
// ... /* ... */

#include, #define,
#ifdef, #ifndef,...



Was Cg/HLSL nicht bereitstellt

- Mehrfachverzweigungen **switch**
- Sprünge **goto**
- Pointer *****
- chars und strings **char, char ***
- Keine rekursiven Funktionen möglich!
- Da C und nicht C++, dürfen natürlich keine objektorientierten Konstrukte verwendet werden (**class, public, private :: etc.**)



Vorgefertigte Konstruktoren



Vorgefertigte Konstruktoren, z.B.:

- float4 f4 = float4(1.0, 3.0, 2.0, 0.5);
- float3 f3Normale1 = float3(4.0, -2.0, 5.0);
- float2 f2UV = float2(0.5, 1.0);

1 // / / /

2 // / / /

3 // / / /

4 // / / /

5 // / / /



Crashkurs in Shadersprachen - Wo Cg/HLSL über C hinauswächst

Swizzling

1 //|//|//|

2 //|//|//|

3 //|//|//|

4 //|//|//|

5 //|//|//|

Vorgefertigte Konstruktoren, z.B.:

- float4 f4 = float4(1.0, 3.0, 2.0, 0.5);
- float3 f3Normale1 = float3(4.0, -2.0, 5.0);
- float2 f2UV = float2(0.5, 1.0);

Swizzling:

- float2 f2 = f3Normale.xy; // vec2 = (4.0, -2.0)
- float frY = f2UV.y; // frY = 1.0
- float3 frColor3 = frY.xxx; // frY = (1.0, 1.0, 1.0)

Möglich sind die Suffixe x,y,z,w oder r,g,b,a



Crashkurs in Shadersprachen - Wo Cg/HLSL über C hinauswächst

Schreibmaskierung

Vorgefertigte Konstruktoren, z.B.:

- float4 f4 = float3(1.0, 3.0, 2.0, 0.5);
- float3 f3Normale1 = float3(4.0, -2.0, 5.0);
- float2 f2UV = float2(0.5, 1.0);

Schreibmaskierung (Write masking):

- float vec4.xw = 7.0; // vec1 = (7.0, 3.0, 2.0, 7.0)

Möglich sind auch hier die Suffixe **x,y,z,w** oder **r,g,b,a**



Discards

- Durch das vorgefertigte Wort `discard` wird die Routine abgebrochen. Anders als bei `return`, welche Cg/HLSL ebenfalls bereitstellt, werden alle bisher berechneten Ergebnisse verworfen.

1 // / / /

2 // / / /

3 // / / /

4 // / / /

5 // / / /



Hardware-Limitierungen

- Cg/HLSL-Programme werden sehr stark von der GPU-Hardware limitiert.
- Die Limitierungen variieren stark von Grafikkarte zu Grafikkarte
- Es gibt hardwareseitige Limitierungen der
 - Registergrößen
 - Arraygrößen
 - Instruktionenanzahl





Crashkurs in Shadersprachen

Übung zu StyleOwn



Erzeugen Sie einen Rotstichfilter!



Freiwillige Zusatzaufgabe für die schnellen Nerds:

Erzeugen Sie ihren eigenen Toon-Shader!



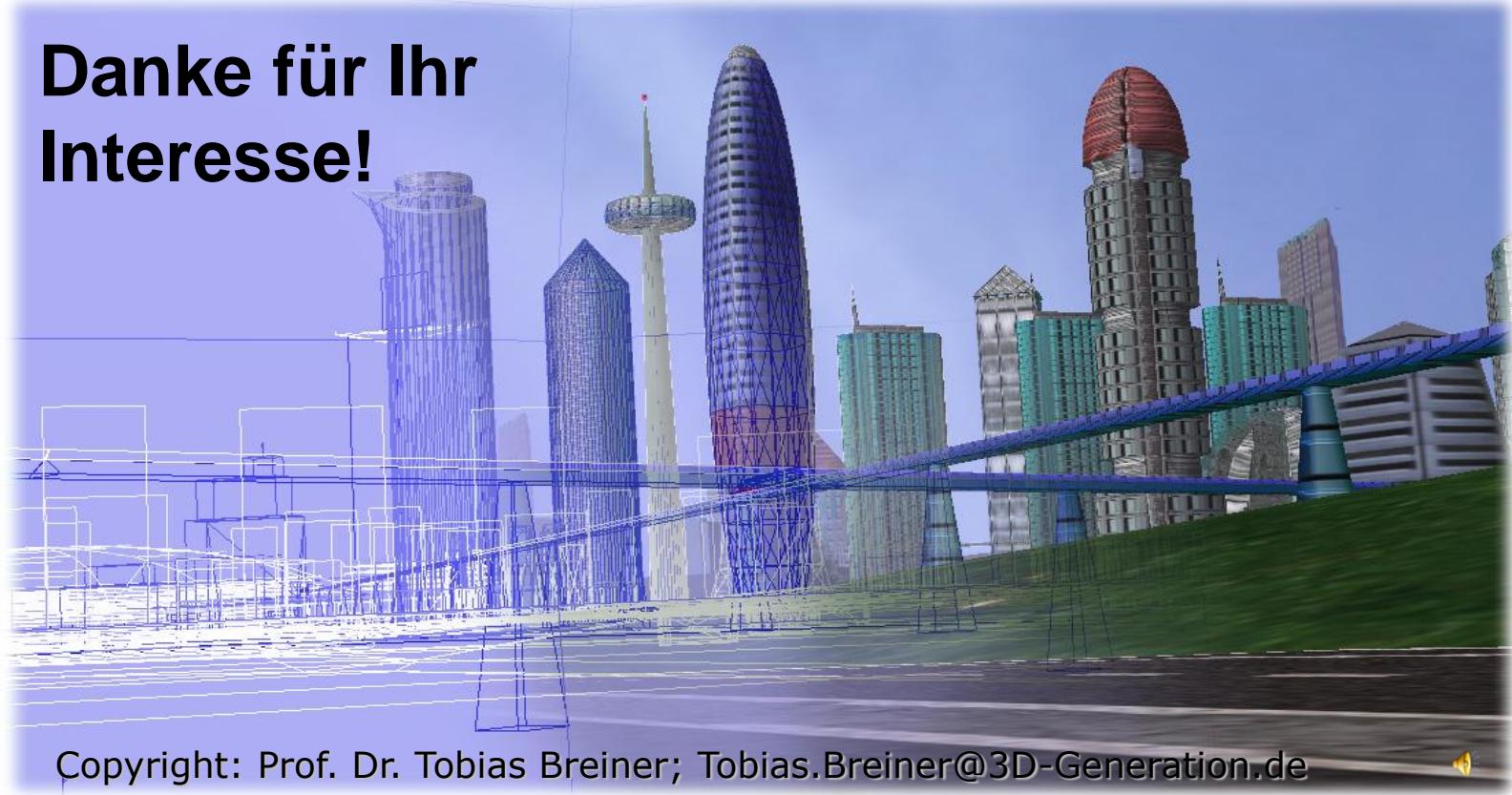
Freiwillige Zusatzaufgabe für die Hardcore-Nerds:

Erzeugen Sie ein stereoskopisches Bild für die Rot-Grün-Brille!



|||||GAME ||||OVER

Danke für Ihr
Interesse!



Copyright: Prof. Dr. Tobias Breiner; Tobias.Breiner@3D-Generation.de

