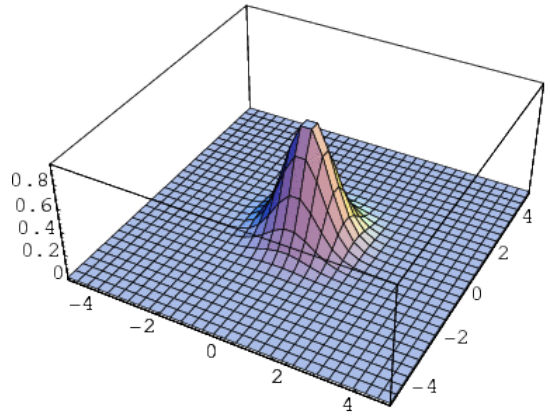
Część IV

Postprocessing – rozmycie gaussowskie

Do tej pory obiekty renderowane były od razu do okna podglądu. Zmienimy trochę ten schemat tworzenia obrazu. Wszystkie przebiegi, które zostały utworzone do tej pory przekierujemy tak, żeby zapisywały obraz do tekstury o wymiarach identycznych z oknem podglądu. Oprócz tej tekstury wykorzystamy dwie dodatkowe tekstury, których każdy wymiar będzie o połowę mniejszy od wymiaru okna podglądu. Te tekstury posłużą nam jako pamięć pomocnicza przy filtrowaniu obrazu zapisanego w dużej teksturze.

Filtr gaussowski jest przykładem filtru macierzowego: wynikowy kolor piksela obrazu powstaje przez pomnożenie wartości koloru pikseli w pewnym kwadracie naokoło niego przez macierz współczynników rzeczywistych.



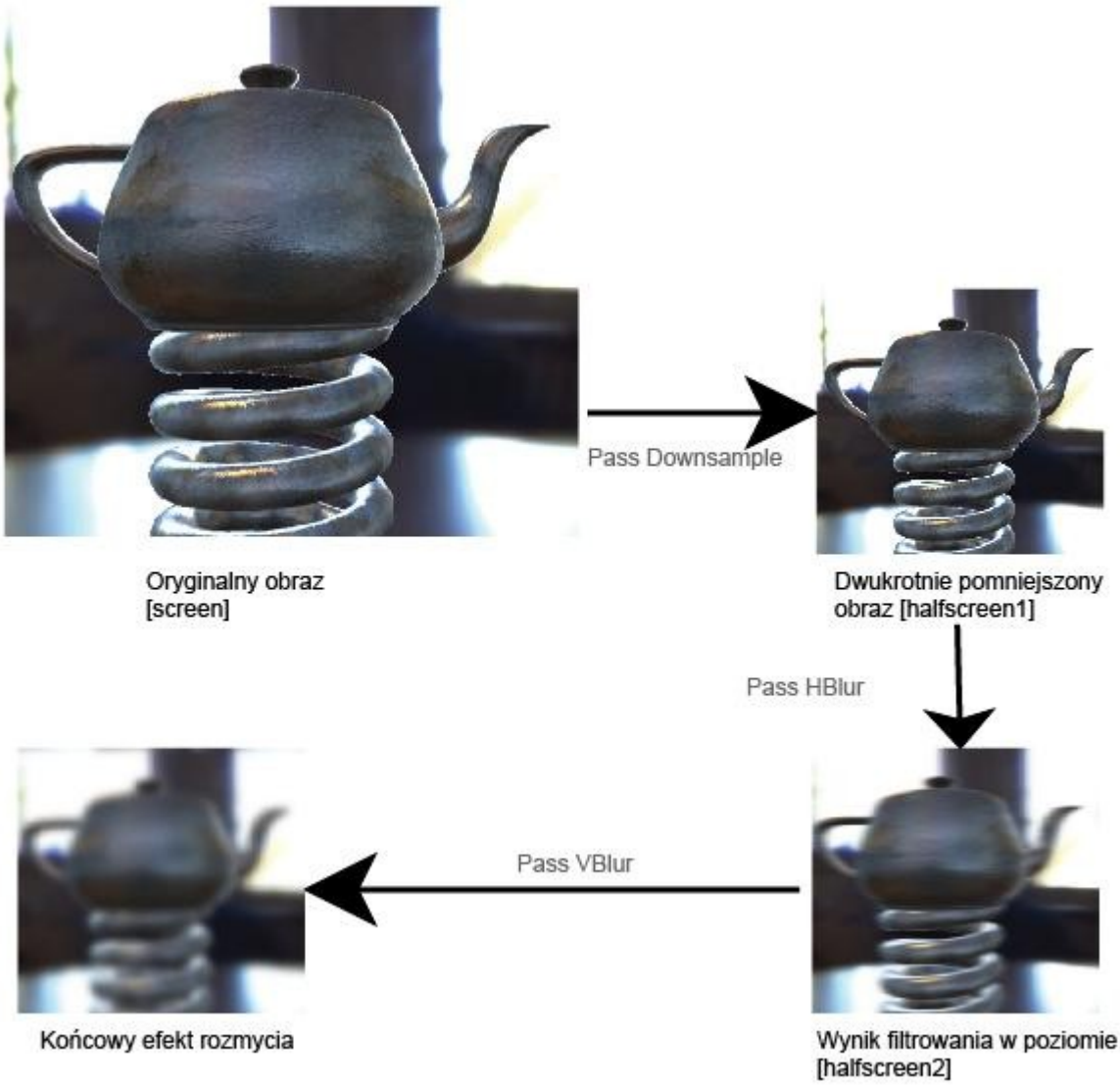
Zaletą rozmycia gaussowskiego jest to, że jest separowalne. Identyczny efekt jak dla macierzy kwadratowej można uzyskać filtrując najpierw obraz w poziomie z odpowiednią macierzą o jednym wierszu, a następnie wynik tej operacji filtrując w pionie z użyciem transpozycji tego wiersza (separowalność polega na tym, że macierz kwadratowa filtru gaussowskiego powstaje z pomnożenia wektora kolumnowego przez identyczny wektor wierszowy).

Postprocessing obrazów przy pomocy shaderów odbywa się z reguły według następującego schematu. Renderowany jest pojedynczy kwadrat (dwa trójkąty), który wypełnia cały obszar okna widoku (tekstury, do której zapisujemy wynik). Obraz, który filtrujemy jest teksturą nałożoną na ten kwadrat. W shaderze pikseli mamy współrzędne tekstury odpowiadające dokładnie pozycji renderowanego piksela. Mając szerokość i wysokość pojedynczego piksela we współrzędnych tekstury, możemy poruszać się po obszarze źródłowego obrazu, odczytywać wartości poszczególnych pikseli i liczyć na ich podstawie wynikowy kolor.

Dwie tekstury pomocnicze mają dwukrotnie mniejsze rozmiary od oryginalnego obrazu. Tak często postępuje się przy rozmywaniu obrazu w grafice czasu rzeczywistego, ponieważ wówczas otrzymuje się dwukrotnie większe rozmycie przy czterokrotnie mniejszej liczbie operacji. Dobrze jest jeszcze dobierać współrzędne tekstury podczas próbkowania tak, aby trafić dokładnie w środek między dwójką (lub czwórką) tekseli. Wtedy w wyniku biliniowego filtrowania sprzętowego dostaniemy średni kolor dwóch (czterech) tekseli zamiast wartości pojedynczego teksela i efekt rozmycia będzie większy.

Dodamy w sumie trzy przebiegi – w każdym renderowany będzie prostokąt wypełniający cały ekran. Pierwszy przebieg pobierze dane z dużej tekstury i zapisze do pierwszej z małych tekstur. Drugi pobierze dane z pierwszej z małych tekstur i zapisze wynik rozmycia w poziomie do drugiej z małych tekstur. Trzeci pobierze dane z drugiej z małych tekstur i wyświetli efekt rozmycia jej w pionie już bezpośrednio na ekranie.

Poniżej pokazane są poszczególne etapy, w nawiasach kwadratowych znajdują się nazwy tekstur, które utworzymy w projekcie, a przy strzałkach nazwy przebiegów renderowania.



1. Możesz wyłączyć animację sprężyny: prawy przycisk na ***time***, **Variable Semantic, Clear Semantic** (ponowne włączenie **Variable Semantic**, **Time0_X**)
2. Dodaj trzy renderowalne tekstury do projektu – *screen* o wymiarach okna widoku oraz *halfscreen1* i *halfscreen2* o wymiarach o połowę mniejszych: prawy przycisk na **Textured Phong, Add Texture, Add Renderable Texture**; prawy przycisk na **renderTexture, Rename, *screen***; dwuklik na ***screen***, zaznacz **Use viewport dimensions**, odznacz **Auto-Generate Mip Map**, kliknij **OK**; powtórz to samo dla ***halfscreen1*** i ***halfscreen2***, ale tym razem dla obu w oknie właściwości, podaj **Viewport Ratio Width = 0.5** oraz **Height = 0.5**
3. Dodaj do projektu parametr, który będzie zawsze przechowywał rozmiar pojedynczego teksela (rozmiar liczony we współrzędnych tekstury) w teksturze *screen*: prawy przycisk na **Textured Phong, Add Variable, Float, Predefined, fInverseViewportDimensions** (i tym razem zostawmy domyślną nazwę)
4. Dodaj do projektu jeszcze jeden parametr związany z wielkością rozmycia *blurscale*: prawy przycisk na **Textured Phong, Add Variable, Float, Float**; prawy przycisk na **f1MyFloat, Rename, *blurscale***; dwuklik na ***blurscale***, wpisz wartość **1.0**, kliknij **OK**; prawy przycisk na ***blurscale***, **Artist Variable**
5. Dodaj siatkę kwadratu złożonego z dwóch trójkątów, którą wykorzystamy w przebiegach związanych z rozmyciem obrazu: prawy przycisk na **Textured Phong, Add Model, Model**; prawy przycisk na **myModel3, rename, Quad**; prawy przycisk na **Quad, Edit**, wskaż plik **ScreenAlignedQuad.3ds** (jest dołączony do materiałów do zajęć umieszczonych na stronie www)
6. Przekieruj renderowanie w przebiegach Pass Cube, Pass Teapot, Pass Spring do tekstury *screen*: prawy przycisk na **Pass Cube, Add Render Target, *screen***; tak samo dla pozostałych dwóch przebiegów
7. Dla wszystkich przebiegów poza pierwszym wyłącz czyszczenie bufora koloru i bufora głębokości: dwuklik na ***screen*** w gałęzi **Pass Teapot**, odznacz **Enable color clear**, odznacz **Enable depth clear**, kliknij **OK**; analogicznie dla **Pass Spring**
8. Dodaj i skonfiguruj nowy przebieg Pass Downsample:
    a. prawy przycisk na **Textured Phong, Add Pass**; prawy przycisk na **Pass 3**, **Rename, Pass Downsample**;
    b. prawy przycisk na **Model** w gałęzi **Pass Downsample, Reference Node, Quad**;
    c. prawy przycisk na **Stream Mapping** w gałęzi **Pass Downsample, Reference Node, Stream Mapping Plane**;
    d. prawy przycisk na **Pass Downsample, Add Texture Object**; prawy przycisk na **Texture0, Rename, src**; rozwiń **src**, prawy przycisk na **baseMap**, **Reference Node, screen**; dwuklik na **src**, ustaw **D3DSAMP_ADDRESSU** i **D3DSAMP_ADDRESSV** na **D3DTADDRESS_CLAMP**
    e. prawy przycisk na **Pass Downsample, Add Render Target, *halfscreen1***
9. Przejdź do edycji shadera wierzchołków w przebiegu Pass Downsample:
    a. Usuń zmienną globalną `float4x4 matViewProjection;` (nie będzie wykorzystywana)
    b. Dodaj do struktury `VS_OUTPUT` pozycję
    ```
    float2 TexCoord : TEXCOORD0;
    ```
    c. Zmień funkcję vs_main na poniższą:

```
VS_OUTPUT vs_main( VS_INPUT In )
{
   VS_OUTPUT Out;
   Out.Position.xy = sign(In.Position);
   Out.Position.z = 1.0;
```

```
    Out.Position.w = 1.0;
    Out.TexCoord.x = Out.Position.x * 0.5 + 0.5;
    Out.TexCoord.y = 1.0 - (Out.Position.y * 0.5 + 0.5);
    return Out;
}
```

10. Przejdź do edycji shadera pikseli w przebiegu Pass Downsample:
    a. Dodaj zmienną globalną skojarzoną z samplerem *src*:
    `sampler2D src;`
    b. Zmień funkcję `ps_main` na poniższą:

```
float4 ps_main( float2 Tex : TEXCOORD0 ) : COLOR0
{
    float4 color  = tex2D(src,Tex);
    return color;
}
```

11. Skopiuj przebieg Pass Downsample jako przebieg Pass HBlur: prawy przycisk na **Pass Downsample, Copy**; prawy przycisk na **Pass Downsample, Paste**; prawy przycisk na **Pass Downsample_1, Rename, Pass HBlur**
12. Skonfiguruj przebieg Pass HBlur tak, żeby czytał z tekstury *halfscreen1* i zapisywał do *halfscreen2*: prawy przycisk na ***halfscreen1*** w gałęzi **Pass HBlur, Reference Node, *halfscreen2***; prawy przycisk; rozwiń ***src*** w gałęzi **Pass HBlur**, prawy przycisk na ***screen***, **Reference Node, *halfscreen1***
13. Przejdź do edycji shadera pikseli w przebiegu Pass HBlur (nie ma potrzeby zmieniania shadera wierzchołków skopiowanego z Pass Downsample):
    a. Dodaj w zasięgu globalnym tablicę wag dla filtra rozmycia:

```
static const float blurweights[13] =
{
  0.002216,
  0.008764,
  0.026995,
  0.064759,
  0.120985,
  0.176033,
  0.199471,
  0.176033,
  0.120985,
  0.064759,
  0.026995,
  0.008764,
  0.002216,
};
```

    b. Dodaj w zasięgu globalnym następujące dwie zmienne powiązane z parametrami projektu:

```
float blurscale;
float2 fInverseViewportDimensions;
```

    c. Usuń wnętrze funkcji `ps_main` i zastąp ją poniższym kodem, który próbkuje teksturę *src* w 13 punktach (6 na lewo, 6 na prawo od pozycji renderowanego piksela, raz w tym samym punkcie) rozmieszczonych w stałych odległościach (równych szerokości teksela pomnożonej przez *blurscale*), mnoży przez wagi tablicy *blurweights* i sumuje:

```
 float4 Color = 0;
 float2 fTexelSize = blurscale*2.0*fInverseViewportDimensions;
 for (int i = 0; i < 13; i++)
     Color += blurweights[i] *
```

```
tex2D( src, Tex + float2(((i-6)*2.0-0.5)*fTexelSize.x,0.0).xy );
    return Color;
```

14. Skopiuj przebieg Pass HBlur jako przebieg Pass VBlur: prawy przycisk na **Pass HBlur, Copy**; prawy przycisk na **Pass HBlur, Paste**; prawy przycisk na **Pass HBlur_1, Rename, Pass VBlur**
15. Skonfiguruj przebieg Pass VBlur tak, żeby czytał z tekstury *halfscreen2* i zapisywał do *halfscreen1* (wynik rozmycia zapisany w teksturze przyda się jeszcze): prawy przycisk na ***halfscreen2*** w gałęzi **Pass VBlur, Reference Node, *halfscreen1***; prawy przycisk; rozwiń ***src*** w gałęzi **Pass VBlur**, prawy przycisk na ***halfscreen1***, **Reference Node, *halfscreen2***
16. Przejdź do edycji shadera pikseli w przebiegu Pass VBlur. Dokonamy tylko jednej małej zmiany – zamienimy poziom z pionem w funkcji `ps_main`:

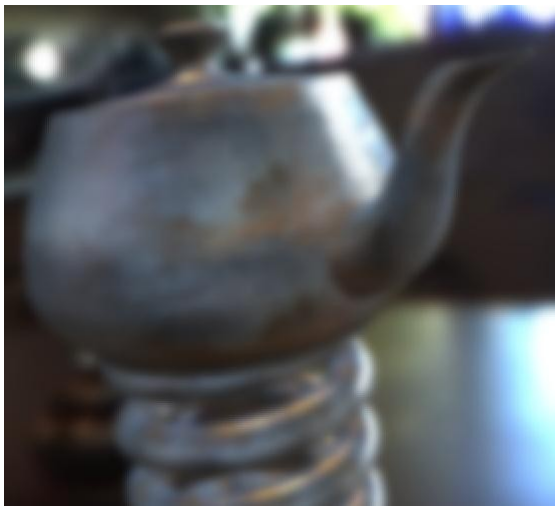
zastąp wyrażenie

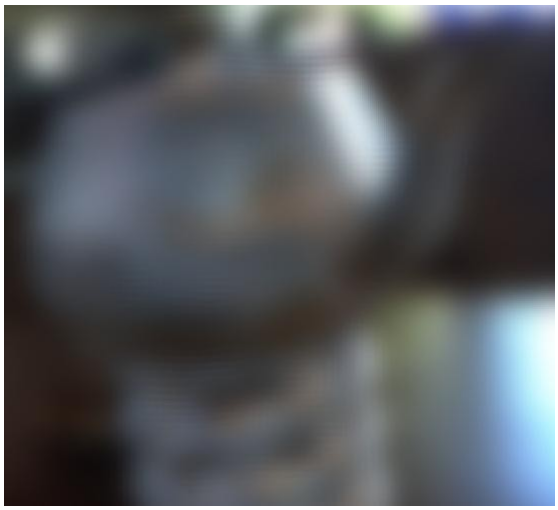
```
float2(((i-6)*2.0-0.5)*fTexelSize.x,0.0).xy
```

wyrażeniem

```
float2(0.0,((i-6)*2.0-0.5)*fTexelSize.y).xy
```

17. Włącz renderowanie wyniku przebiegu Pass VBlur na ekran: prawy przycisk na ***halfscreen1*** w gałęzi **Pass VBlur, Render To Screen**
18. Skompiluj shadery (okno podglądu musi być otwarte), otwórz Artist Editor i zobacz jak zmiana parametru blurscale wpływa na renderowany obraz.





Na koniec nieznacznie zmodyfikujemy przebieg Pass Downsample tak, żeby zostawić tylko najjaśniejsze fragmenty obraz, a wynik rozmycia dodamy w nowym przebiegu Pass Composite do pierwotnego obrazu tak, żeby uzyskać poświatę wokół najjaśniejszych fragmentów.

19. Dodaj nowy parametr luminance_cutoff do projektu: prawy przycisk na **Textured Phong, Add Variable, Float, Float**; prawy przycisk na **f1MyFloat, Rename, luminance_cutoff**; dwuklik na **luminance_cutoff**, wartość **2.16**, kliknij **OK**; prawy przycisk na **luminance_cutoff, Artist Variable**
20. Dodaj zmienną globalną do shadera pikseli w przebiegu Pass Downsample:
```
float luminance_cutoff;
```
21. Zmień treść funkcji `ps_main` w przebiegu Pass Downsample:
```
   return dot(color.xyz,float3(1.0,1.0,1.0)) < luminance_cutoff ?
float4(.0,.0,.0,.0) : color;
```
zamiast
```
return color;
```
22. Skopiuj przebieg Pass Downsample jako nowy, ostatni na liście, przebieg o nazwie Pass Composition: prawy przycisk na **Pass Downsample, Copy**; prawy przycisk na **Pass VBlur, Paste**; prawy przycisk na **Pass Downsample_1, Rename, Pass Composition**
23. Usuń renderowanie do tekstury w przebiegu Pass Composition: prawy przycisk na ***halfscreen1*** w gałęzi **Pass Composition, Delete**
24. Dodaj w przebiegu Pass Composition sampler tekstury powiązany z teksturą *halfscreen1*: prawy przycisk na **Pass Composition, Add Texture Object**; prawy przycisk na **Texture1, Rename, src2**; rozwiń **src2**, prawy przycisk na **baseMap, Reference Node, *halfscreen1***
25. Przejdź do edycji shadera pikseli w przebiegu Pass Composition:
    a. Dodaj zmienną globalną powiązaną z samplerem src2:
    ```
    sampler2D src2;
    ```
    b. Usuń całą treść funkcji ps_main i zastąp ją następującą instrukcją:
    ```
    return tex2D(src2,Tex)+tex2D(src,Tex);
    ```

Możesz teraz otworzyć Artist Editor i obserwować w oknie podglądu wynik zmian parametrów luminance_cutoff, exposure i blurscale (te trzy powinny być oznaczone jako Artist Variable).