

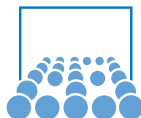
HPCS'2015

# A Runtime/Memory Trade-off of the Continuous Ziggurat Method on GPUs

Christoph Riesinger, Tobias Neckel

Technische Universität München

July 21, 2015



# Topics

## Motivation/Introduction

### The Ziggurat method

- Definition of the Ziggurat
- Algorithmic description

## Memory/runtime trade-off

## Results

- Influence of the number of strips
- Comparison with alternative PRNGs

## Conclusion

# Topics

## Motivation/Introduction

### The Ziggurat method

Definition of the Ziggurat  
Algorithmic description

### Memory/runtime trade-off

### Results

Influence of the number of strips  
Comparison with alternative PRNGs

### Conclusion

## Motivation/Introduction

### Random numbers are required in many fields

- Cryptography
- Monte Carlo methods
- Simulating stochastic processes
- Stochastic/Random differential equations
- ...

## Motivation/Introduction

### Random numbers are required in many fields

- Cryptography
- Monte Carlo methods
- Simulating stochastic processes
- Stochastic/Random differential equations
- ...

### Generation of random numbers on a computer

- Determined by a deterministic rule
- Such random numbers are not really “random” but satisfy certain statistical criteria (*pseudo random numbers*)
- In general, such rules determine a uniformly distributed random number  $u_{integer}^{uniform} \in \{0, \dots, \text{ULONG\_MAX}\}$  and  $u_{float}^{uniform} \in [0, 1[$ , resp.
- A transformation function transforms this number to desired distribution, e.g. normal distribution  $u^{normal}$ , by “imitating” the inverse CDF

⇒ Ziggurat method [MT00]

# Topics

## Motivation/Introduction

## The Ziggurat method

- Definition of the Ziggurat
- Algorithmic description

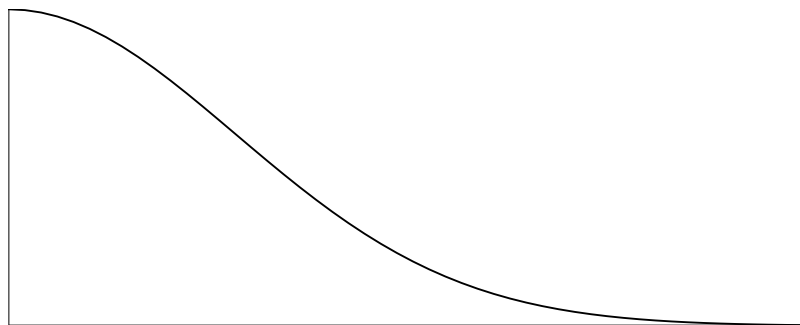
## Memory/runtime trade-off

## Results

- Influence of the number of strips
- Comparison with alternative PRNGs

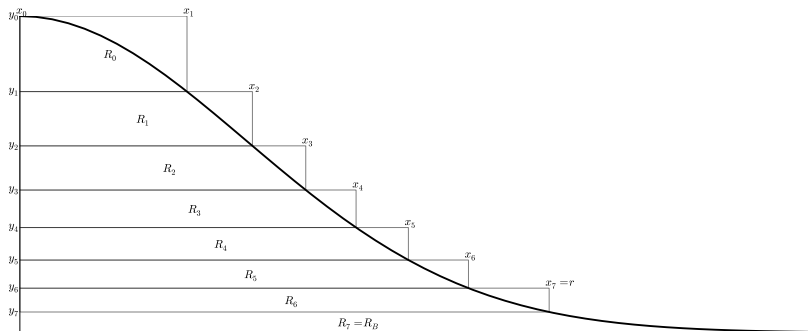
## Conclusion

## Definition of the Ziggurat: Gaussian bell function



$$f(x) = e^{-\frac{x^2}{2}}$$

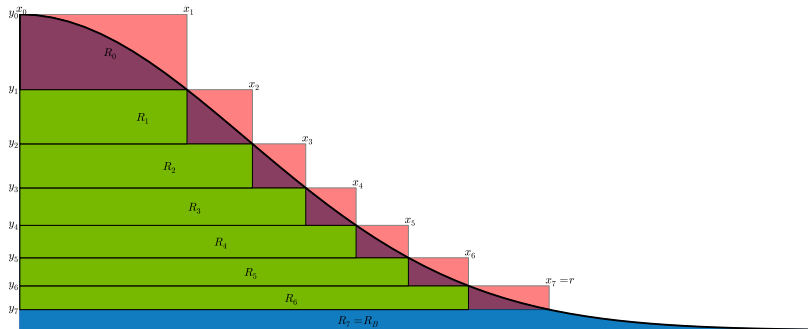
## Definition of the Ziggurat: Approximation by strips



- $R_i$  strip
- $N$  number of strips
- $(x_i, y_i)$  right bottom corner of all rectangular strips  $R_0, \dots, R_{N-2}$
- $v$  common area of all strips  $R_0, \dots, R_{N-1}$
- $r = x_{N-1}$  right-most edge of a rectangular strip



## Definition of the Ziggurat: Four different regions



central region

tail region

cap region

base strip

rectangle totally under bell function

remainder of strip under bell function

remainder of strip over bell function

non-rectangular strip  $R_{N-1}$

## Algorithmic description

The Ziggurat is stored by saving  $x_1, \dots, x_{N-1}$  in a lookup table

$$x_1 = 0.738368917976448$$

$$x_2 = 1.027386371780228$$

$$x_3 = 1.262970198530832$$

$$x_4 = 1.485358675643293$$

$$x_5 = 1.716508125776779$$

$$x_6 = 1.981904936400510$$

$$x_7 = 2.338371698247252$$

## Algorithmic description

The Ziggurat is stored by saving  $x_1, \dots, x_{N-1}$  in a lookup table

$$x_1 = 0.738368917976448$$

$$x_2 = 1.027386371780228$$

$$x_3 = 1.262970198530832$$

$$x_4 = 1.485358675643293$$

$$x_5 = 1.716508125776779$$

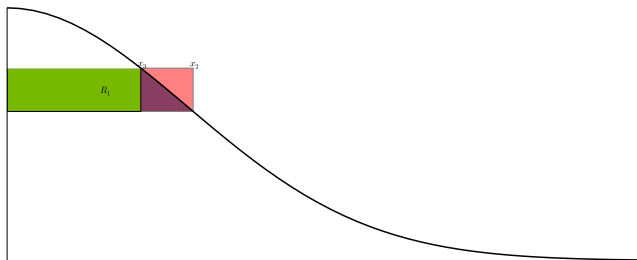
$$x_6 = 1.981904936400510$$

$$x_7 = 2.338371698247252$$

### How the Ziggurat method works

1. Generate a  $u_{integer}^{uniform}$  and its corresponding  $u_{float}^{uniform}$
2. To transform  $u^{uniform}$  to  $u^{normal}$   
select the  $k$ -th strip of the Ziggurat by  $k = u_{integer}^{uniform} \&(2^N - 1)$
3. Depending on  $k$ , the transformation is done as follows. . .

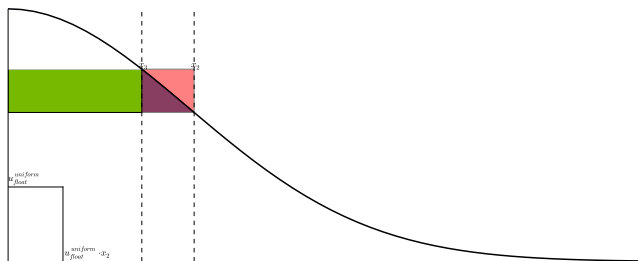
## Algorithmic description: Central region



### A central region is hit if

- $k \neq N - 1$
- $u_{float}^{uniform} \leq x_{k+1}/x_k$

## Algorithmic description: Central region



### A central region is hit if

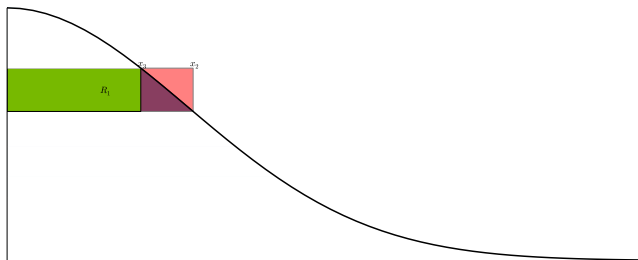
- $k \neq N - 1$
- $u_{float}^{uniform} \leq x_{k+1} / x_k$

### Then, transformation is done by

$$u^{normal} = u_{float}^{uniform} \cdot x_{k+1}$$

$\Rightarrow$  great, because cheap

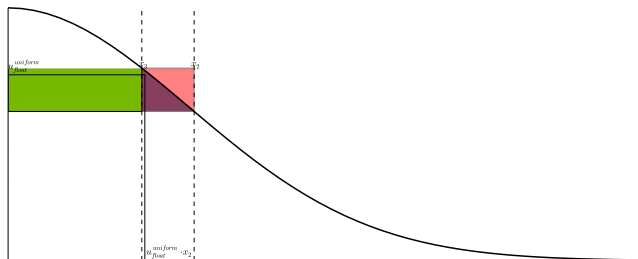
## Algorithmic description: Tail region



### A tail region is hit if

- $k \neq N - 1$
- the central region is not hit
- $$u_{float}^{uniform} \cdot (f(x_{k+1}) - f(x_k)) < f(u_{float}^{uniform} \cdot x_{k+1}) - f(x_{k+1})$$

## Algorithmic description: Tail region



### A tail region is hit if

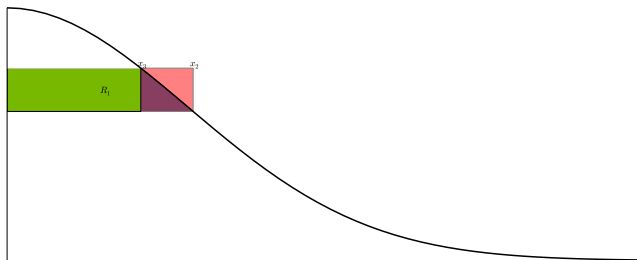
- $k \neq N - 1$
- the central region is not hit
- $$u_{float}^{uniform} \cdot (f(x_{k+1}) - f(x_k)) < f(u_{float}^{uniform} \cdot x_{k+1}) - f(x_{k+1})$$

### Then, transformation is done by

$$u^{normal} = u_{float}^{uniform} \cdot x_{k+1}$$

$\Rightarrow$  great, because cheap

## Algorithmic description: Cap region

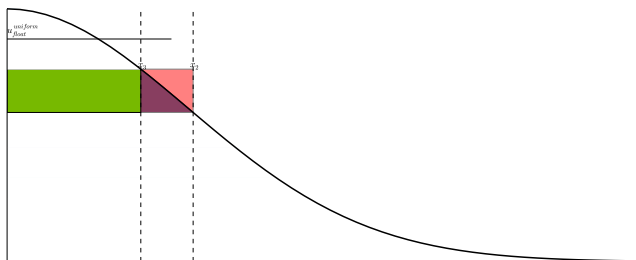


### A cap region is hit if

- $k \neq N - 1$
- neither the central nor tail region are hit



## Algorithmic description: Cap region



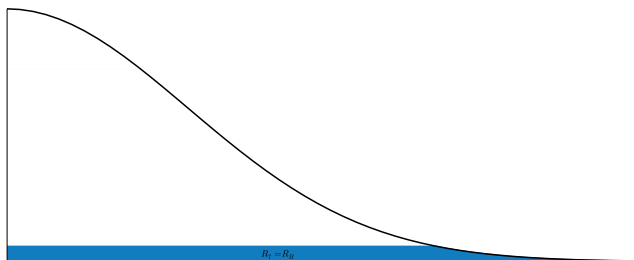
### A cap region is hit if

- $k \neq N - 1$
- neither the central nor tail region are hit

### Then, transformation is done by

Restart the Ziggurat method with new  $u^{uniform}$   
 $\Rightarrow$  **bad**, because expensive

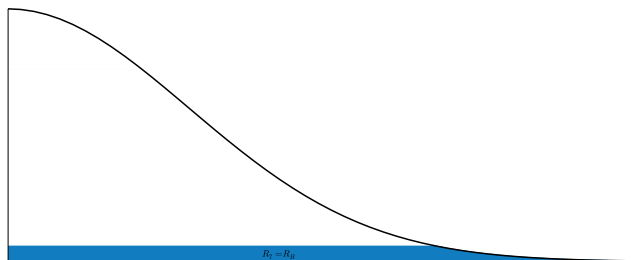
## Algorithmic description: Base strip



The *base strip* is hit if

- $k = N - 1$

## Algorithmic description: Base strip



### The *base strip* is hit if

- $k = N - 1$

### Then, transformation is done by

$$x = (v \cdot u_{float}^{uniform}) / f(r)$$

- If  $x < r$ , then  $u^{normal} = x$
- If  $x \geq r$ , then [Mar64]

⇒ **bad**, because expensive

# Topics

## Motivation/Introduction

## The Ziggurat method

Definition of the Ziggurat  
Algorithmic description

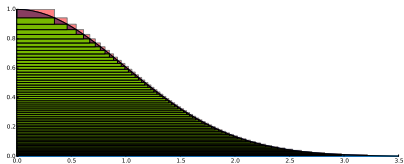
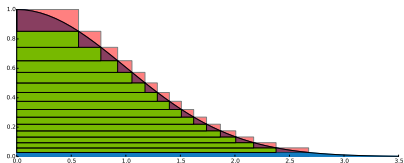
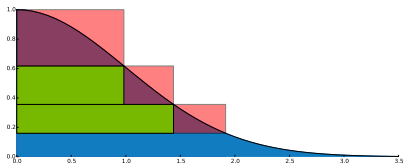
## Memory/runtime trade-off

## Results

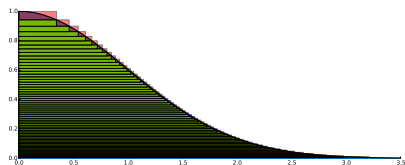
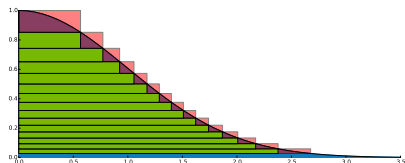
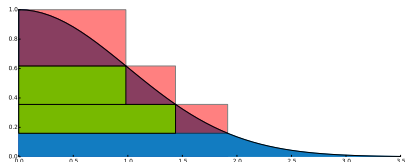
Influence of the number of strips  
Comparison with alternative PRNGs

## Conclusion

# Influence of the number of strips



# Influence of the number of strips



Number of strips	% of covered area
$2 = 2^1$	0.0%
$4 = 2^2$	~ 42.98%
$8 = 2^3$	~ 69.75%
$16 = 2^4$	~ 84.13%
$32 = 2^5$	~ 91.71%
$64 = 2^6$	~ 95.64%
$128 = 2^7$	~ 97.71%
$256 = 2^8$	~ 98.80%
$512 = 2^9$	~ 99.37%
$1024 = 2^{10}$	~ 99.67%
$2048 = 2^{11}$	~ 99.83%
$4096 = 2^{12}$	~ 99.91%
$8192 = 2^{13}$	~ 99.95%

## HPC aspects of number of used strips

- The more strips are used for the Ziggurat, the bigger the ratio of the sum of all central regions to the sum of all strips gets
- The bigger this ratio gets, the higher the likelihood to hit a (cheap) central region gets
- In addition, on **GPUs**, this reduces the likelihood for warp divergence

## HPC aspects of number of used strips

- The more strips are used for the Ziggurat, the bigger the ratio of the sum of all central regions to the sum of all strips gets
- The bigger this ratio gets, the higher the likelihood to hit a (cheap) central region gets
- In addition, on **GPUs**, this reduces the likelihood for warp divergence

#strips	likelihood of warp div.
$2 = 2^1$	$1 - 0.0^{32} = 100.0\%$
$4 = 2^2$	$1 - 0.4298^{32} \approx 99.9\%$
$8 = 2^3$	$1 - 0.6975^{32} \approx 99.9\%$
$16 = 2^4$	$1 - 0.8413^{32} \approx 99.9\%$
$32 = 2^5$	$1 - 0.9171^{32} \approx 93.7\%$
$64 = 2^6$	$1 - 0.9564^{32} \approx 76.0\%$
$128 = 2^7$	$1 - 0.9771^{32} \approx 52.3\%$
$256 = 2^8$	$1 - 0.9880^{32} \approx 32.0\%$
$512 = 2^9$	$1 - 0.9937^{32} \approx 18.3\%$
$1024 = 2^{10}$	$1 - 0.9967^{32} \approx 10.0\%$
$2048 = 2^{11}$	$1 - 0.9983^{32} \approx 5.3\%$
$4096 = 2^{12}$	$1 - 0.9991^{32} \approx 2.8\%$
$8192 = 2^{13}$	$1 - 0.9995^{32} \approx 1.6\%$



## HPC aspects of number of used strips

- The more strips are used for the Ziggurat, the bigger the ratio of the sum of all central regions to the sum of all strips gets
- The bigger this ratio gets, the higher the likelihood to hit a (cheap) central region gets
- In addition, on **GPUs**, this reduces the likelihood for warp divergence
- So runtime can be reduced by using more strips which results in larger lookup tables  
⇒ runtime/memory trade-off

#strips	likelihood of warp div.
$2 = 2^1$	$1 - 0.0^{32} = 100.0\%$
$4 = 2^2$	$1 - 0.4298^{32} \approx 99.9\%$
$8 = 2^3$	$1 - 0.6975^{32} \approx 99.9\%$
$16 = 2^4$	$1 - 0.8413^{32} \approx 99.9\%$
$32 = 2^5$	$1 - 0.9171^{32} \approx 93.7\%$
$64 = 2^6$	$1 - 0.9564^{32} \approx 76.0\%$
$128 = 2^7$	$1 - 0.9771^{32} \approx 52.3\%$
$256 = 2^8$	$1 - 0.9880^{32} \approx 32.0\%$
$512 = 2^9$	$1 - 0.9937^{32} \approx 18.3\%$
$1024 = 2^{10}$	$1 - 0.9967^{32} \approx 10.0\%$
$2048 = 2^{11}$	$1 - 0.9983^{32} \approx 5.3\%$
$4096 = 2^{12}$	$1 - 0.9991^{32} \approx 2.8\%$
$8192 = 2^{13}$	$1 - 0.9995^{32} \approx 1.6\%$

# Topics

## Motivation/Introduction

## The Ziggurat method

Definition of the Ziggurat  
Algorithmic description

## Memory/runtime trade-off

## Results

Influence of the number of strips  
Comparison with alternative PRNGs

## Conclusion

# Setup

## GPUs

GPU architecture	Fermi	Kepler	Maxwell
Model	M2090	Tesla K40m	GTX 750 Ti
Compute capability	2.0	3.5	5.0
#Processing elements	$16 \times 32$	$15 \times 192$	$5 \times 128$
Size of shared memory (KB)	48	48	48
SP peak performance (TFLOPS)	1.3312	3.84192	1.6384

# Setup

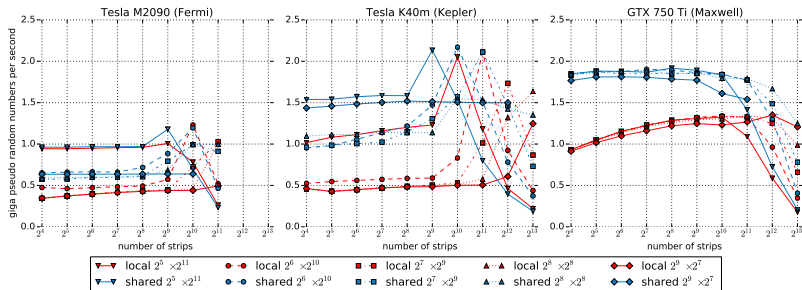
## GPUs

GPU architecture	Fermi	Kepler	Maxwell
Model	M2090	Tesla K40m	GTX 750 Ti
Compute capability	2.0	3.5	5.0
#Processing elements	$16 \times 32$	$15 \times 192$	$5 \times 128$
Size of shared memory (KB)	48	48	48
SP peak performance (TFLOPS)	1.3312	3.84192	1.6384

## Setup

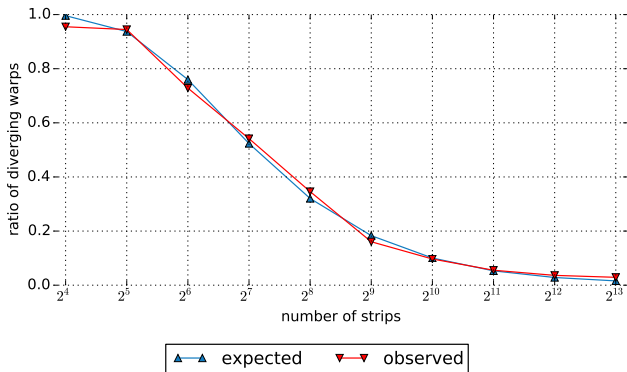
- For every run,  $2^{28}$  float numbers are generated
- Each thread produces  $2^{12}$  random numbers
- All floating point operations are done in single precision
- Uniform input  $u_{integer}^{uniform}$  is generated by cuRAND XORWOW [Mar03]
- Execution times contain generation of  $u_{integer}^{uniform}$  and time for transformation

# Influence of the number of strips



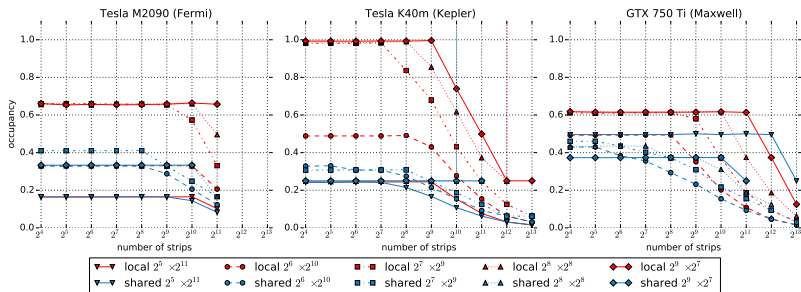
- States for XORWOW can be stored in **local** or **shared** memory
- Value of interest is giga pseudo random numbers per second (GPRNs/s)
- For most configurations, a single peak of performance is observable

## Influence of the number of strips: Warp divergence



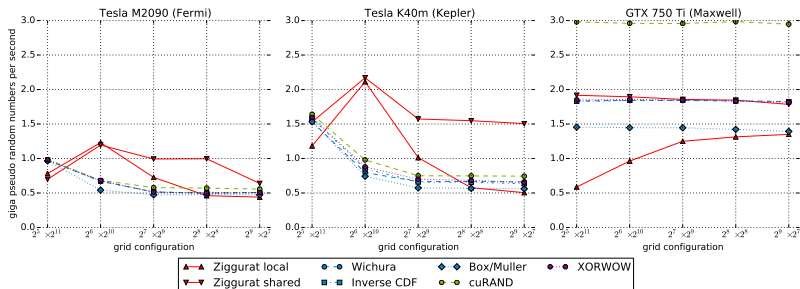
⇒ For low strip numbers, performance is limited by high warp divergence

# Influence of the number of strips: Occupancy



⇒ For large strip numbers, performance is limited by low occupancy

## Comparison with alternative PRNGs



- For Ziggurat, best configuration (in terms of used strips) is used
- *Wichura* [Wic88] is a direct generator for random numbers, *inverse CDF* uses `normcdfinvf()`, and *Box/Muller* [BM58] is the most popular transformation function
- Ziggurat shows better performance than all other methods on Fermi (+19.9%) and Kepler (+24.3%), but not on Maxwell (-35.8%)



# Topics

## Motivation/Introduction

## The Ziggurat method

Definition of the Ziggurat  
Algorithmic description

## Memory/runtime trade-off

## Results






Influence of the number of strips  
Comparison with alternative PRNGs

## Conclusion

## Conclusion

- The Ziggurat method is a rejection method to transform uniformly distributed random numbers to normally distributed random numbers
- A runtime/memory trade-off allows reducing the execution time of the transformation by spending more memory for larger lookup tables
- Especially GPUs benefit from this trade-off since it leads to lower warp divergence
- While an implementation for CPUs typically uses 128 or 256 strips, a flat recommendation for GPUs is not possible
- Our Ziggurat implementation is up to 19.9% faster on Fermi and up to 24.3% faster on Kepler

## Bibliography

-  George Edward Pelham Box and Mervin Edgar Muller.  
A Note on the Generation of Random Normal Deviates.  
*The Annals of Mathematical Statistics*, 29(2):610–611, 1958.
-  George Marsaglia.  
Generating a variable from the tail of the normal distribution.  
*Technometrics*, 6(1):101–102, 1964.
-  George Marsaglia.  
Xorshift RNGs.  
*Journal of Statistical Software*, 8:1–6, 2003.
-  George Marsaglia and Wai Wan Tsang.  
The Ziggurat Method for Generating Random Variables.  
*Journal of Statistical Software*, 5(8):1–7, 2000.
-  Michael J. Wichura.  
Algorithm AS241: The percentage points of the normal distribution.  
*Applied Statistics*, 37:477–484, 1988.

# Final slide

