RenderScript Atomic Update Functions

Overview

To update values shared between multiple threads, use the functions below. They ensure that the values are atomically updated, i.e. that the memory reads, the updates, and the memory writes are done in the right order.

These functions are slower than their non-atomic equivalents, so use them only when synchronization is needed.

Note that in RenderScript, your code is likely to be running in separate threads even though you did not explicitely create them. The RenderScript runtime will very often split the execution of one kernel across multiple threads. Updating globals should be done with atomic functions. If possible, modify your algorithm to avoid them altogether.

Summary

Functions	
rsAtomicAdd	Thread-safe addition
rsAtomicAnd	Thread-safe bitwise and
rsAtomicCas	Thread-safe compare and set
rsAtomicDec	Thread-safe decrement
rsAtomicInc	Thread-safe increment
rsAtomicMax	Thread-safe maximum
rsAtomicMin	Thread-safe minimum
rsAtomicOr	Thread-safe bitwise or
rsAtomicSub	Thread-safe subtraction
rsAtomicXor	Thread-safe bitwise exclusive or

Functions

rsAtomicAdd : Thread-safe addition

int32_t rsAtomicAdd(volatile int32_t* addr, int32_t value);	Added in API level 14
int32_t rsAtomicAdd(volatile uint32_t* addr, uint32_t value);	Added in API level 20

Parameters

addr	Address of the value to modify.
value	Amount to add.

Returns

Value of *addr prior to the operation.

Atomicly adds a value to the value at addr, i.e. `*addr += value`.

rsAtomicAnd : Thread-safe bitwise and

int32_t rsAtomicAnd(volatile int32_t* addr, int32_t value);	Added in API level 14

int32_t rsAtomicAnd(volatile uint32_t* addr, uint32_t value);   Added in API level 20

Parameters

addr	Address of the value to modify.
value	Value to and with.

Returns

Value of *addr prior to the operation.

Atomicly performs a bitwise and of two values, storing the result back at addr, i.e. `*addr &= value`.

rsAtomicCas : Thread-safe compare and set

int32_t rsAtomicCas(volatile int32_t* addr, int32_t compareValue, int32_t newValue);   Added in API level 14

uint32_t rsAtomicCas(volatile uint32_t* addr, uint32_t compareValue, uint32_t newValue);   Added in API level 14

Parameters

addr	Address of the value to compare and replace if the test passes.
compareValue	Value to test *addr against.
newValue	Value to write if the test passes.

Returns

Value of *addr prior to the operation.

If the value at addr matches compareValue then the newValue is written at addr, i.e. `if (*addr == compareValue) { *addr = newValue; }`.

You can check that the value was written by checking that the value returned by rsAtomicCas() is compareValue.

rsAtomicDec : Thread-safe decrement

int32_t rsAtomicDec(volatile int32_t* addr);   Added in API level 14

int32_t rsAtomicDec(volatile uint32_t* addr);   Added in API level 20

Parameters

addr	Address of the value to decrement.

Returns

Value of *addr prior to the operation.

Atomicly subtracts one from the value at addr. This is equivalent to `rsAtomicSub(addr, 1)`.

rsAtomicInc : Thread-safe increment

int32_t rsAtomicInc(volatile int32_t* addr);   Added in API level 14

int32_t rsAtomicInc(volatile uint32_t* addr);   Added in API level 20

Parameters

addr	Address of the value to increment.

Returns

Value of *addr prior to the operation.

Atomicly adds one to the value at addr. This is equivalent to `rsAtomicAdd(addr, 1)`.

rsAtomicMax : Thread-safe maximum

int32_t rsAtomicMax(volatile int32_t* addr, int32_t value);   Added in API level 14

uint32_t rsAtomicMax(volatile uint32_t* addr, uint32_t value);   Added in API level 14

Parameters

addr	Address of the value to modify.
value	Comparison value.

Returns

Value of *addr prior to the operation.

Atomicly sets the value at addr to the maximum of *addr and value, i.e. `*addr = max(*addr, value)`.

rsAtomicMin : Thread-safe minimum

int32_t rsAtomicMin(volatile int32_t* addr, int32_t value);	Added in API level 14
uint32_t rsAtomicMin(volatile uint32_t* addr, uint32_t value);	Added in API level 14

Parameters

addr	Address of the value to modify.
value	Comparison value.

Returns

Value of *addr prior to the operation.

Atomicly sets the value at addr to the minimum of *addr and value, i.e. `*addr = min(*addr, value)`.

rsAtomicOr : Thread-safe bitwise or

int32_t rsAtomicOr(volatile int32_t* addr, int32_t value);	Added in API level 14
int32_t rsAtomicOr(volatile uint32_t* addr, uint32_t value);	Added in API level 20

Parameters

addr	Address of the value to modify.
value	Value to or with.

Returns

Value of *addr prior to the operation.

Atomicly perform a bitwise or two values, storing the result at addr, i.e. `*addr |= value`.

rsAtomicSub : Thread-safe subtraction

int32_t rsAtomicSub(volatile int32_t* addr, int32_t value);	Added in API level 14
int32_t rsAtomicSub(volatile uint32_t* addr, uint32_t value);	Added in API level 20

Parameters

addr	Address of the value to modify.
value	Amount to subtract.

Returns

Value of *addr prior to the operation.

Atomicly subtracts a value from the value at addr, i.e. `*addr -= value`.

rsAtomicXor : Thread-safe bitwise exclusive or

int32_t rsAtomicXor(volatile int32_t* addr, int32_t value);	Added in API level 14
int32_t rsAtomicXor(volatile uint32_t* addr, uint32_t value);	Added in API level 20

Parameters

addr	Address of the value to modify.
value	Value to xor with.

Returns

Value of *addr prior to the operation.

Atomicly performs a bitwise xor of two values, storing the result at addr, i.e. `*addr ^= value`.